3-Phase BLDC Sensorless Control with MQX RTOS Using the K60N512

Document Number: DRM135
Rev. 0, 06/2012

freescale™

Contents

Chapter 1
Introduction

1.1 Introduction

This design reference manual (DRM) describes the design of a sensorless 3-phase brushless DC (BLDC) motor drive based on Freescale's 32-bit Kinetis K60 device.

BLDC motors are very popular in a wide application area. The BLDC motor lacks a commutator and is therefore more reliable than the DC motor. The BLDC motor also has advantages when compared to an AC induction motor. Because it achieves a higher efficiency by generating the rotor magnetic flux with rotor magnets, a BLDC motor is used in high-end white goods, such as refrigerators, washing machines, dishwashers, high-end pumps, fans, and in other appliances that require a high reliability and efficiency.

The concept of the application is a speed closed-loop BLDC drive using a sensorless BEMF zero-crossing technique. It serves as an example of a BLDC motor control design using a Freescale K60 device MCU. It is focused on a simple and "easy to understand" control approach to BLDC, and using MQX in a time critical application. The application contains two versions of the application software. One is with the MQX RTOS, and the other is bare-metal. Both use the same source code for motor control. The MQX version contains a web server to demonstrate the benefits of an MQX-based solution.

This reference design includes a basic motor theory, the system design concept, hardware implementation, and software design, including the FreeMASTER software visualization tool.

The hardware is built on the Freescale Tower rapid prototyping system and contains the following modules:

- TWR-Elevator
- TWR-K60N512
- TWR-MC-LV3PH
- TWR-SER

This design shows the advantages of the motor control peripherals of the Kinetis microcontrollers. The control algorithm will include:

- start-up with rotor alignment
- sensorless position detection using integration of Back-EMF voltage
- speed closed-loop
- current limitation
- fault protection

1.2 Freescale K60 advantages and features

The 32-bit Kinetis MCUs represent the most scalable portfolio of ARM® Cortex™-M4 MCUs in the industry. The first phase of the portfolio consists of five MCU families with over 200 pin-, peripheral-, and software-compatible devices with outstanding performance, memory, and feature scalability. Enabled by innovative 90 nm Thin Film Storage flash technology with unique FlexMemory, Kinetis features the latest low-power innovations and high-performance, high-precision mixed-signal capability. Kinetis MCUs are supported by a market-leading enablement bundle from Freescale and ARM® third party ecosystem partners.

The K60 MCU family includes IEEE 1588 Ethernet, full- and high-speed USB 2.0 On-The-Go with device charge detect capability, hardware encryption and tamper detection capabilities. Devices start from 256 KB of flash in 100-pin LQFP packages, extending up to 1 MB in a 256-pin MAPBGA package with a rich suite of analogue, communication, timing, and control peripherals.

High memory density K60 family devices include an optional single precision floating point unit, NAND flash controller, and DRAM controller.

Freescale Tower System hardware development environment:

- Integrated development environments:
  - Eclipse-based CodeWarrior V10.x IDE and Processor Expert
  - IAR Embedded Workbench
  - Keil MDK
  - CodeSourcery Sourcery G++ (GNU)
- Runtime software and RTOS:
  - Maths, DSP, and encryption libraries
  - Motor control libraries
  - Complimentary bootloaders (USB, Ethernet, RF, serial)
  - Complimentary Freescale embedded GUI
  - Complimentary Freescale MQX™
  - Cost-effective NanoSSL™ /NanoSSH™ for Freescale MQX RTOS

- Micrium uC/OS-III
- Express Logic ThreadX
- SEGGER embOS
- freeRTOS,
- Mocana (security)

- Full ARM ecosystem

Chapter 2
Control Theory

2.1 Brushless DC motor (BLDC motor)

A BLDC motor is a rotating electric machine where the stator is a classic 3-phase stator like that of an induction motor, and the rotor has surface-mounted permanent magnets, see Figure 2-1. The same arrangement is used in the Linix 45ZWN24-40.

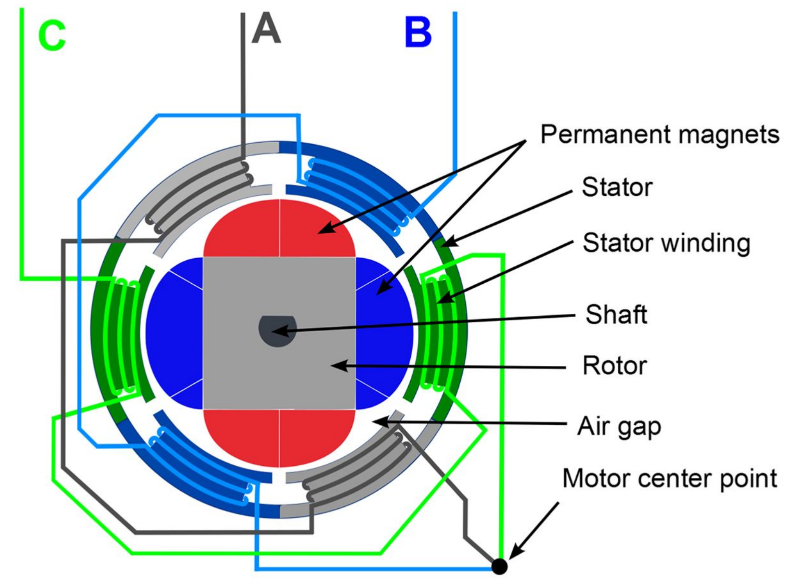


Figure 2-1. BLDC motor – cross section

2.2 Digital control of a BLDC motor

The BLDC motor is driven by rectangular voltage strokes coupled with the given rotor position, see Figure 2-2. The generated stator flux interacts with the rotor flux, which is generated by a rotor magnet, defines the torque and thus the speed of the motor. The voltage strokes must be properly applied to the two phases of the 3-phase winding system so that the angle between the stator flux and the rotor flux is kept close to 90°, to get the maximum generated torque. Because of this fact, the motor requires electronic control for proper operation.

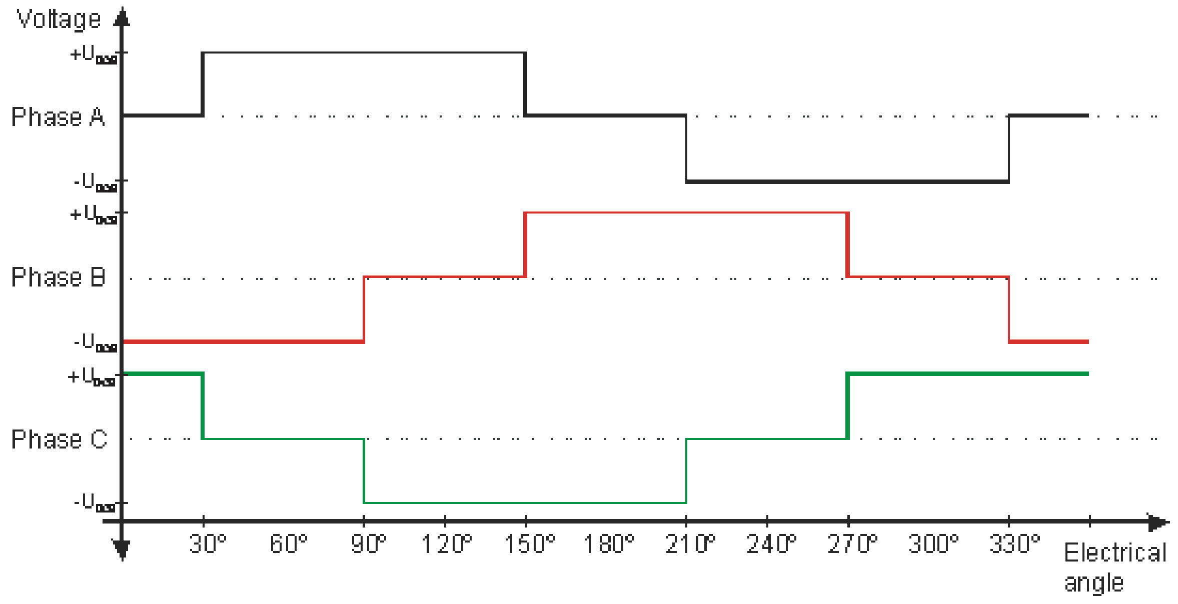


Figure 2-2. Voltage strokes applied to the 3-phase BLDC motor

For the common 3-phase BLDC motor, a standard 3-phase power stage is used, as is illustrated in Figure 2-3. The power stage utilizes six power transistors.

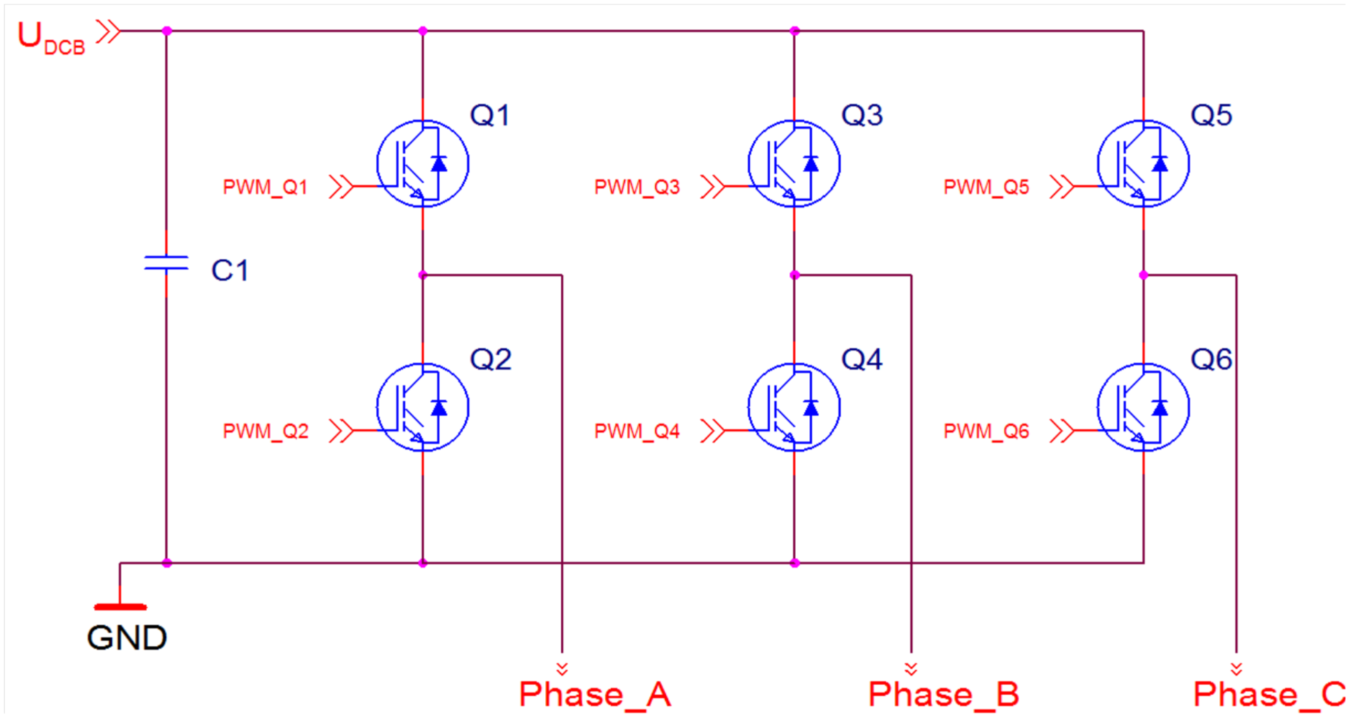


Figure 2-3. 3-phase BLDC power stage

In both modes, the 3-phase power stage energizes two motor phases concurrently. The third phase is unpowered, see Figure 2-2. Thus, we get six possible voltage vectors that are applied to the BLDC motor using a PWM technique. There are two basic types of power transistor switching: independent switching and complementary switching.

2.3 Complementary versus independent switching

With complementary switching, two transistors are switched on when the phase of the BLDC motor is connected to the power supply. But there is a difference during freewheeling. With independent switching, all the transistors are switched off and the current continues to flow in the same direction through freewheeling diodes until it falls to zero. Contrary to this, with complementary switching, the complementary transistors are switched on during freewheeling. Thus, the current may be able to flow in the opposite direction. Figure 2-4 depicts the complementary switching.

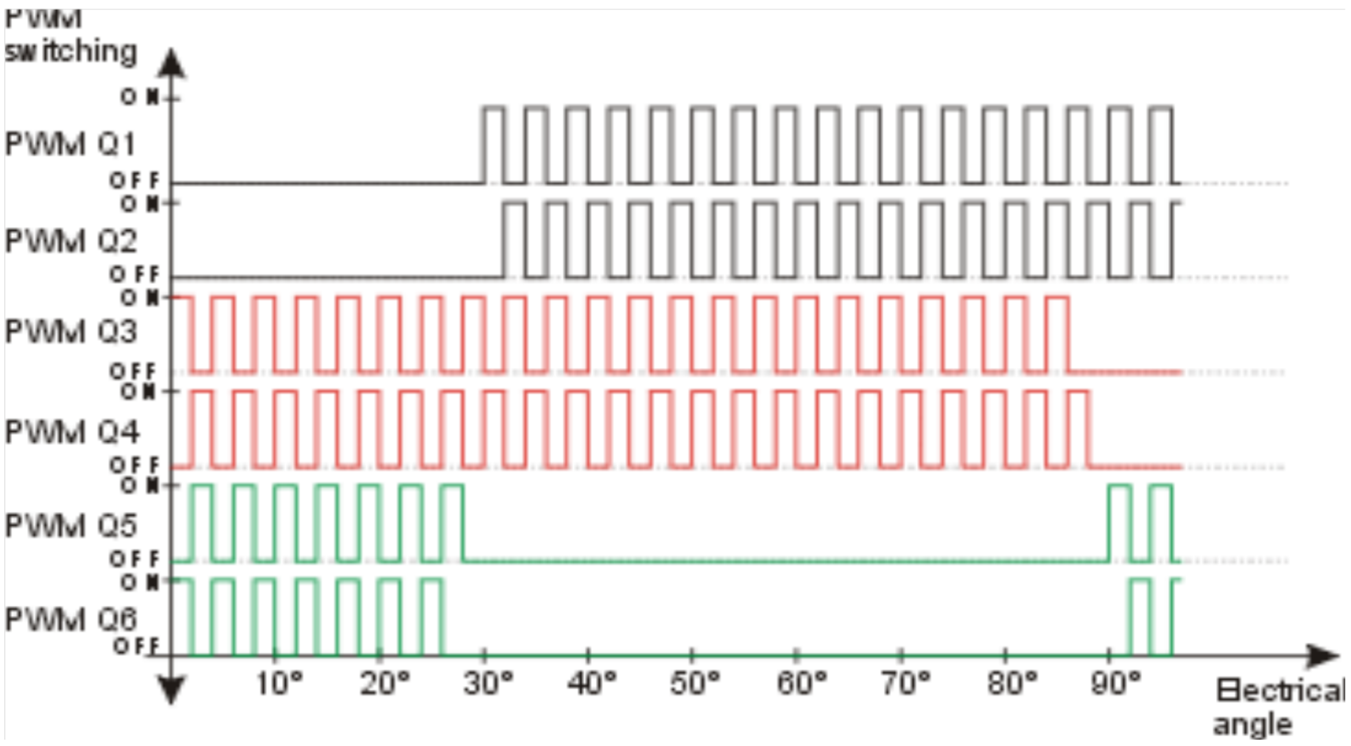


Figure 2-4. Complementary switching of power transistors

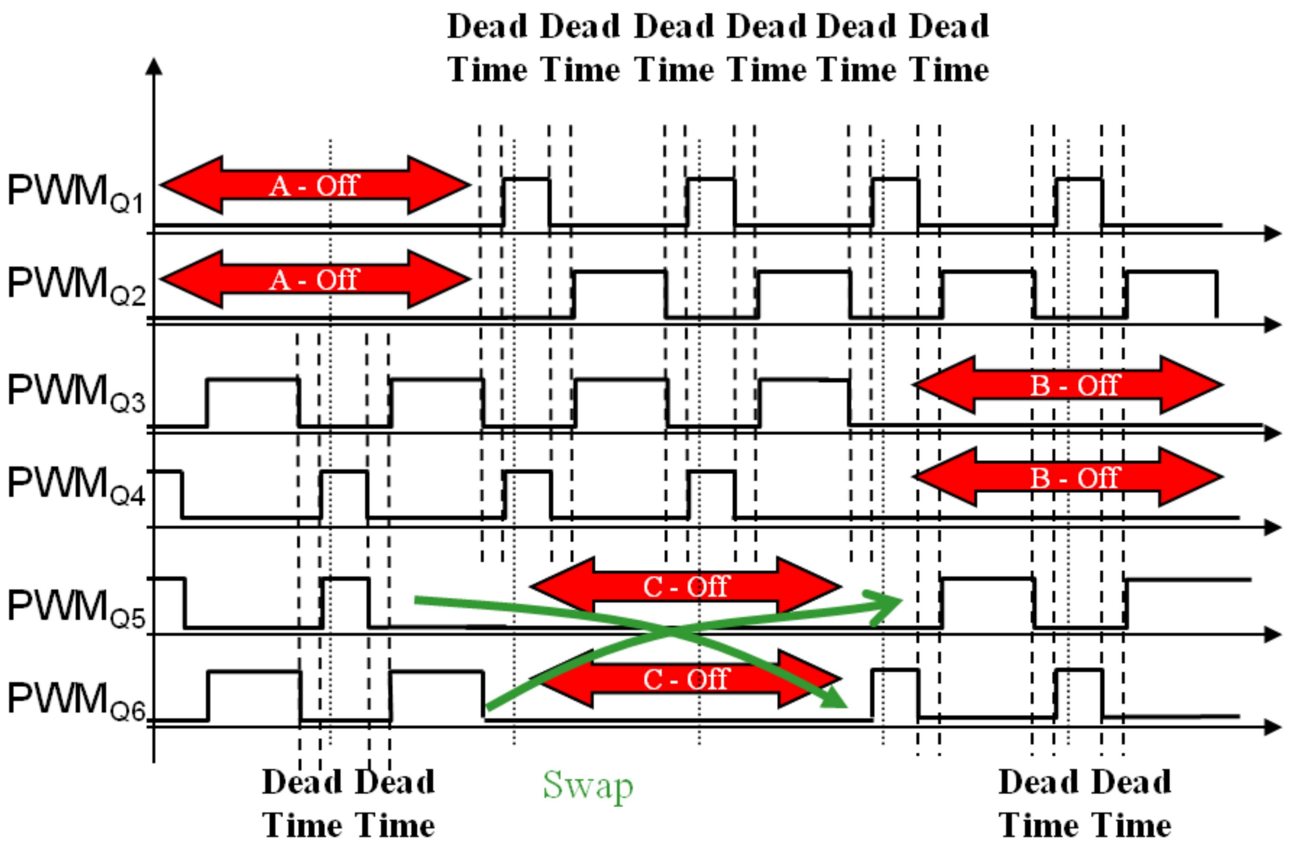


Figure 2-5. Bipolar PWM switching — detail

Details of the technique are shown in Figure 2-5. From Figure 2-5 we can see the characteristic of the bipolar 4-quadrant complementary switching. The bipolar switching requires that the top and bottom switch PWM signals need to be swapped. Another important detail is the introduction of dead time insertion in the complementary top and bottom signals. This dead time insertion is typical for all 4-quadrant power stage operations. The 4-quadrant operation is enabled by the complementary operation of the top and bottom switches. The bottom switch of one phase is almost the negative of the top switch.

This requires the insertion of a dead time because the switching transient will cause a DC-bus short circuit with fatal power stage damage.

The bipolar PWM switching is not as popular as the unipolar switching because of a worse electromagnetic emission of the motor. This is because the PWM ripple is twice that of the DC-bus voltage. On the other hand, this switching is better for sensorless rotor position sensing.

Details are described in the documentation, see References .

2.3.1 The 4-quadrant operation

As described in the previous sections, the amplitude of a 3-phase voltage system needs to be controlled. The most common BLDC control topology uses the power stage with a constant POWER SOURCE DC VOLTAGE. Therefore, the 3-phase average voltage amplitude is controlled by a PWM technique of the top and bottom transistors. As described, the 6-step controller uses one of the two PWM techniques:

1. Bipolar PWM switching
2. Unipolar PWM switching

There are a few derivatives of the two PWM switching techniques, according to the operating quadrants of the power stage voltage and current:

1. 4-quadrant power stage control
2. 2-quadrant power stage control

The 2-quadrant operation provides a defined voltage and current of the same polarity, that is, positive voltage with positive current, or negative voltage and negative current, which are the operating quadrants I and III in Figure 2-6.

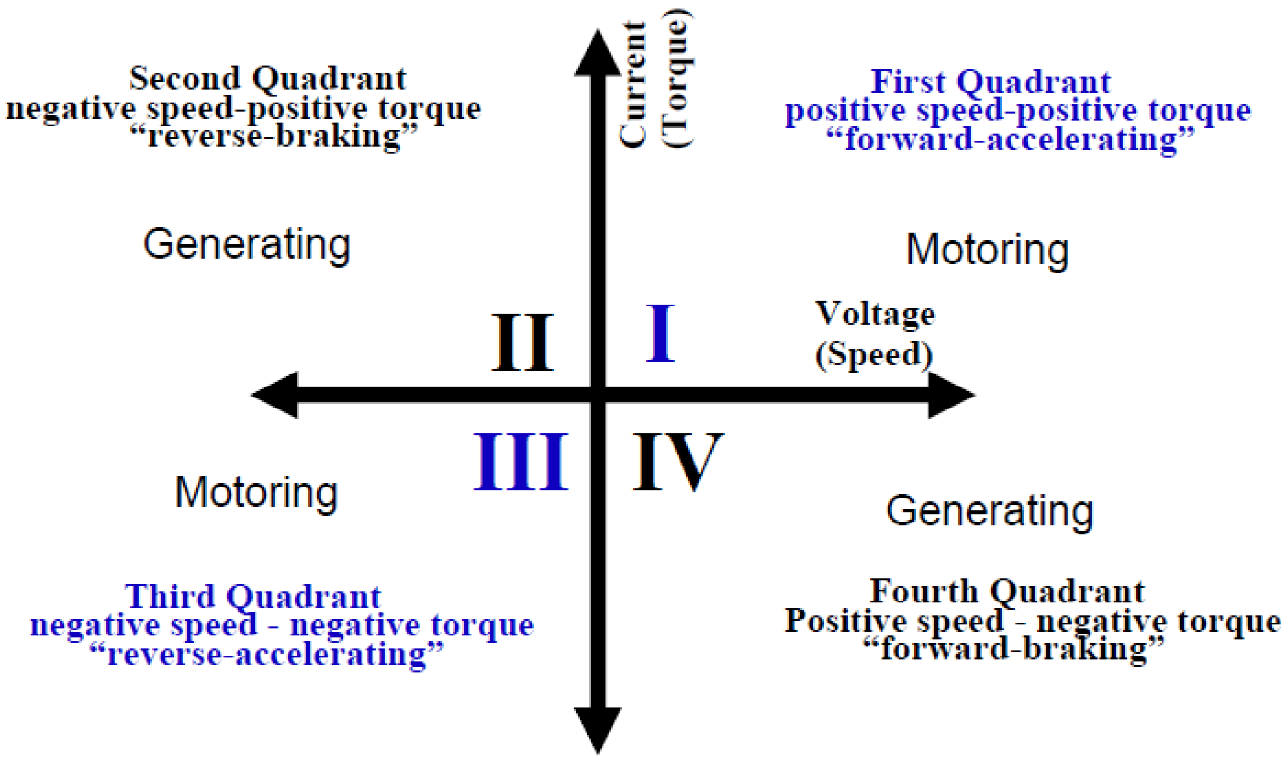


Figure 2-6. The 4-quadrant operation

The 4-quadrant PWM switching covers the operation of the generated voltage in all four quadrants. This is provided using complementary switching of the top and bottom transistors. The benefits of the 4-quadrant PWM operation are:

- Motoring and generating mode control (possibility of braking the motor).
- Linear operation in all four quadrants.

The K60N512 is able to control the 3-phase power stage with unipolar and bipolar PWM, with the more advanced 4-quadrant operation. Therefore, the 4-quadrant PWM technique will be used in this application.

2.4 Mathematical description of a brushless DC motor

2.4.1 Power stage - motor system model

In order to explain the idea of the Back-EMF sensing technique, the basic circuit topology, see Figure 2-7.

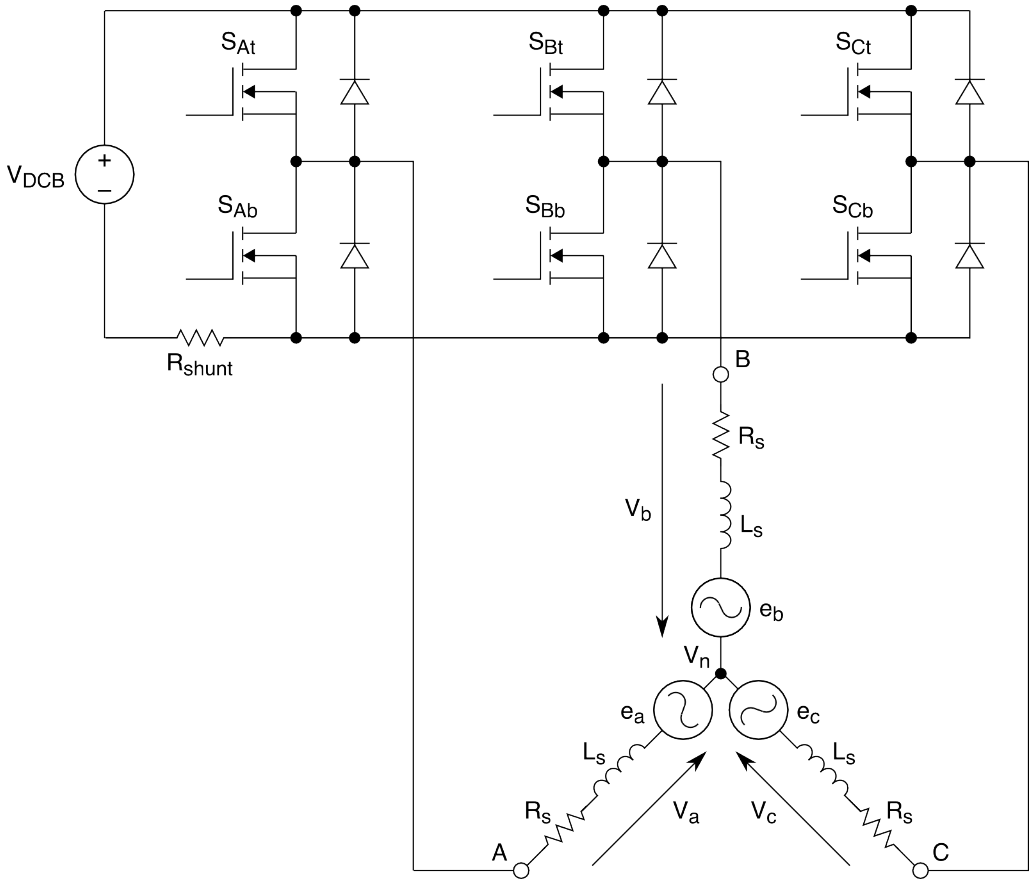


Figure 2-7. Power stage and motor topology

The motor-drive model consists of a 3-phase power stage plus a brushless DC motor. The power for the system is provided by a voltage source (VDCB). Six semiconductor switches (SA/B/C t/b), controlled elsewhere, allow the rectangular voltage waveforms (see Figure 2-2) to be applied. The semiconductor switches and diodes are considered as ideal switches.

2.4.2 Back-EMF sensing

The Back-EMF sensing technique is based on the fact that only two phases of a brushless DC motor are energized at a time. The third phase is a non-fed phase that can be used to sense the Back-EMF voltage.

The Figure 2-8 shows branch and motor phase winding voltages during a 0–360° electrical interval. The yellow interval means a conduction interval of a phase. During this time, current flows through the winding and BEMF voltage is impossible to measure. After the commutation transient, there is a current recirculation. At this time, the fly-back diodes conduct the decaying phase current. Blue lines determine the time when the Back-EMF voltage can be sensed during the designated intervals. Green lines determine the time when the zero-crossing detection can be enabled. The red line shows when the BEMF voltage is integrated, and at the end of the red interval there is the next commutation.

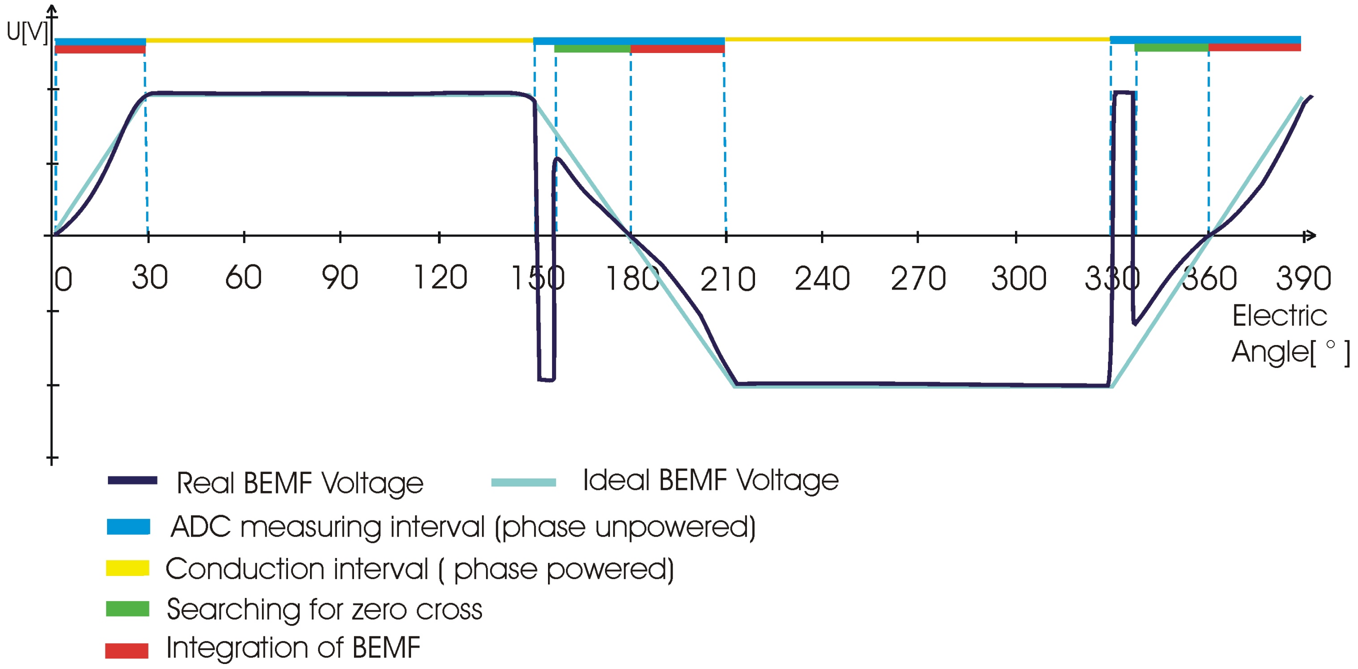


Figure 2-8. Single phase voltage waveform

For more detailed information about BEMF sensing, see References .

Chapter 3
Motor Control With The MXQ RTOS

3.1 What is MQX

Freescale's MQX Software Solutions offer a straightforward API with a modular architecture, making it simple to fine tune custom applications, and scalable to fit most requirements. The combination of market-proven Freescale MQX Software Solutions and silicon portfolio provides a streamlined and powerful platform by creating a comprehensive source for hardware, software, tools, and services needs.

3.2 When to use motor control with the MQX RTOS

MQX is not a typical operating system for motor control applications. The MQX OS is suitable for large applications with advanced features, such as web control, USB, displays and SDHC card reading, running on a dedicated device (usually one core). The primary strength of MQX is that it includes libraries for RTCS, Ethernet, USB communication, the MFS file system, and many other applications.

3.3 How to implement motor control and MQX

The MQX RTOS is a complex system with dynamic allocations and POSIX scheduling. It has a system default tick duration of 5 ms. This is ideal for the majority of applications. However, this also means that the MQX task time resolution is more than 1000 times longer when compared to the motor control requirements. Therefore, it is evident that the motor control process needs to be serviced with interrupts of a high priority. This can be provided with standard MQX interrupt routines. The time duration from the interrupt request to the service routines execution is usually units of a microsecond, depending on the CPU version and clock speed. And, if necessary, the motor control algorithms can be implemented using the kernel interrupts. The kernel interrupts are natural CPU interrupts with no MQX overhead and minimal execution duration. The disadvantage of the kernel interrupt is that no MQX functionalities such as events or semaphores are supported.

Chapter 4
System Concept

4.1 Sensorless drive concept

The concept of this system is described below in the block diagram Figure 4-1.

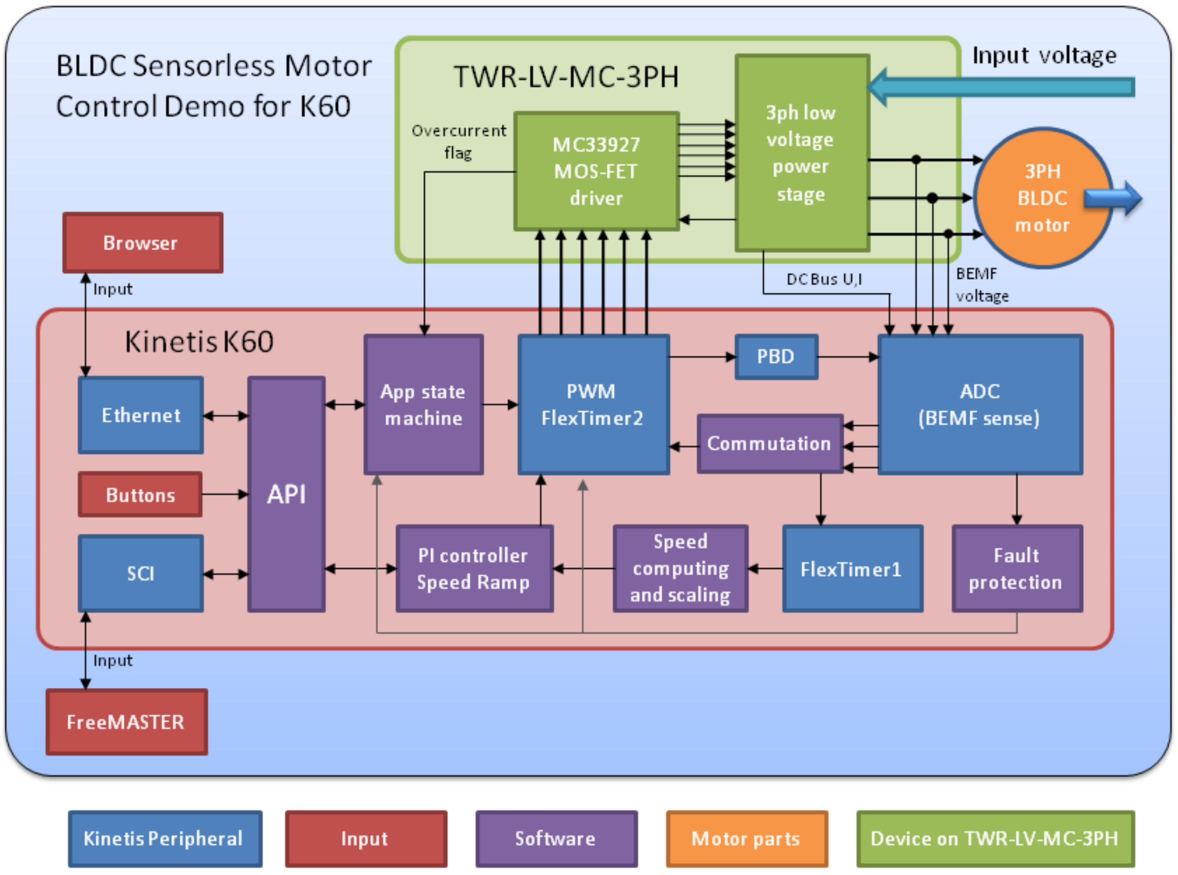


Figure 4-1. Block diagram of system the concept

The basic state machine operation of the system is as follows:

The periodic interrupt timer service routine executes the **application state machine** (ASM), consisting of these states:

- stop
- alignment
- run
- emergency stop
- overcurrent
- overvoltage
- undervoltage

After an MCU reset, the application goes through the MCU_init function into the stop state. The change of the *speed_scaled* is continuously monitored. As soon as the user enters a value other than zero into *speed_req*, the ASM switches the state to alignment. The alignment state performs the alignment in two sectors to avoid a bad alignment. The duration of alignment depends on the value of ALIGNMENT_CYCLE macro in the BLDC.h. This value is in PIT timer cycles. The default value is 6, which means the duration of the alignment is about 6 ms. The applied voltage can be changed using parameter START_DUTY_CYCLE in BLDC_config.h. This value is a percentage of input voltage.

After the motor is in a known position, the next sector is applied to the motor according to the desired motor direction. Then the ASM goes into the run state, where zero-crossing is employed and the control loop is closed. During the run state, requests for new speeds and torques are accepted and maintained by the PI controllers. If a standstill of the rotor is detected, the ASM goes again into the alignment state and tries to execute a new start-up sequence. In the case of any fault, the ASM goes into the error state. The ZC detection process compares the actual BEMF voltage (*phase_bemf*) with half of the DC-bus voltage (*half_dc_bus*). The appropriate phase voltage vector is chosen based on the sector (*sector*). The ZC detection process defines the start interval of BEMF integration.

The required value of the speed (*speed_req*) is updated, based on commands from the user. The required speed slope is limited by a ramp to avoid exceeding the maximal motor current. The speed controller output, together with the current limit, defines the duty cycle (*duty_cycle*) for the FTM/PWM module. The integral part of the PI controller is limited if the overcurrent flag from the MC33937 driver is triggered.

Once a fault condition occurs, the application switches to the Fault state, stops the motor, and waits for a fault reset via the Reset switch on the TWR-K60N512 board. Beyond this, the FreeMASTER interface enables monitoring and adjustment of all system variables.

4.2 System blocks concept

4.2.1 PWM voltage generation for a brushless DC motor

A 3-phase voltage system as described in Brushless DC motor (BLDC motor) needs to be created to run the BLDC motor. This is provided by a 3-phase power stage with six power switches (IGBTs or MOSFETs) controlled by the Kinetis K60 on-chip FTM/PWM module. When generating PWM signals for the BLDC motor control application, the Bottom and Top power switches of the non-fed phase must be switched off (See Figure 2-5). For the BLDC motor control application, PWM signals can be created in two ways: Complementary PWM mode and Independent PWM mode. Both these modes are available in Kinetis K60 FTM/PWM module.

In complementary PWM mode, the top and bottom switches of a phase are operated complementarily. This mode has to be used if four quadrant drive operation is required. This mode needs dead time insertion between the top and bottom switches to avoid any phase short circuit. The complementary switching can be implemented in both a bipolar or unipolar manner. The unipolar switching leads to lower switching losses and current ripple. However, from a Back-EMF point of view, the bipolar switching is a better choice since this allows having a duty cycle in the range of 50 –100 %. This significantly simplifies the Back-EMF voltage and current sensing.

The complementary bipolar switching is selected for this sensorless BLDC drive because of easy implementation of the Back-EMF voltage sensing method and the Kinetis K60 features point of view. The details of the Back-EMF voltage sensing method can be seen in ADC to PWM synchronization.

4.3 ADC to PWM synchronization

The power stage PWM switching causes voltage spikes on the phase voltages. These voltage spikes are generated on the non-fed phase because of mutual inductances and mutual capacitor couplings between the motor windings. Non-fed phase “branch” voltage is then disturbed by PWM switching. Figure 4-2 and Figure 4-3 show the effect on the non-fed phase because of the mutual inductance.

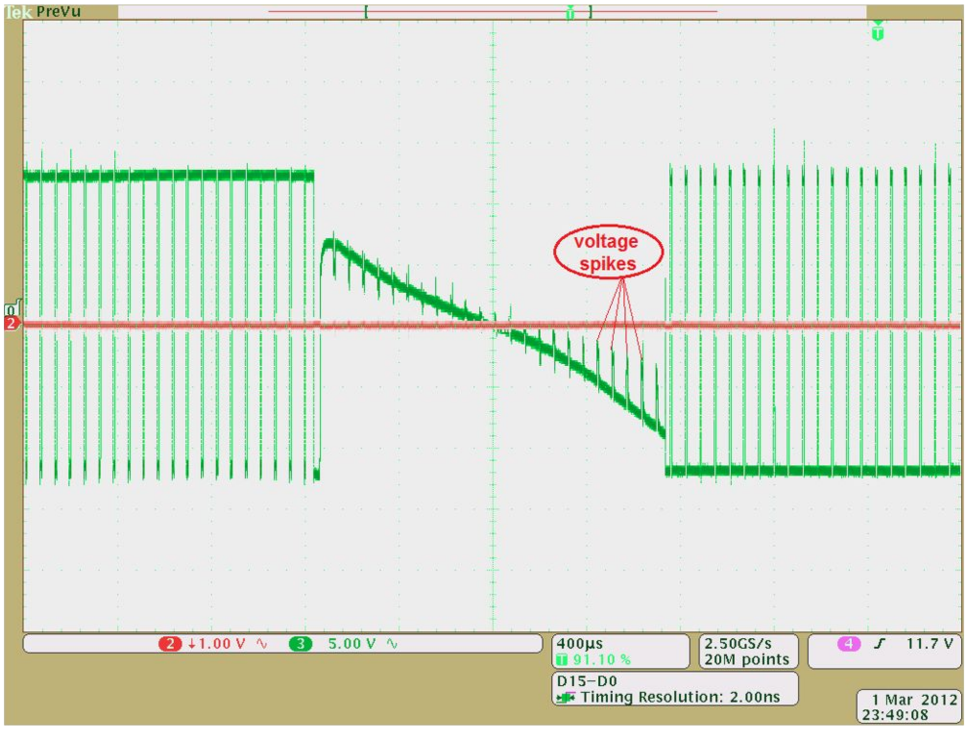


Figure 4-2. BEMF voltage spikes and ADC triggering - A

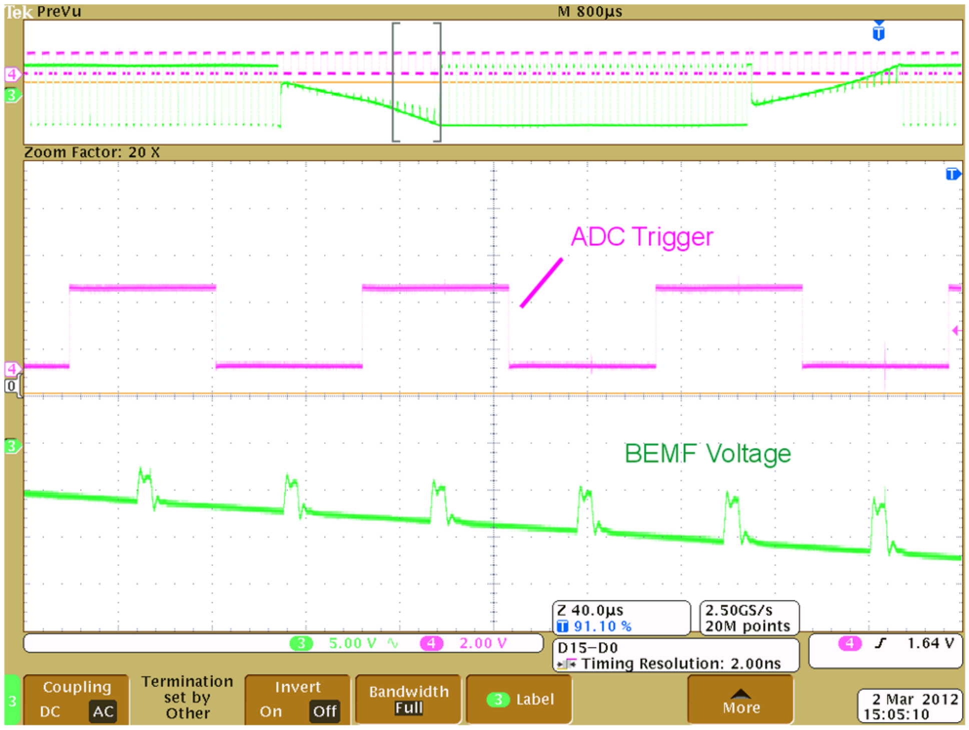


Figure 4-3. BEMF voltage spikes and ADC triggering - B

The non-fed phase “branch” voltage is disturbed at the PWM switching edges. Therefore, the presented BLDC motor control application synchronizes the Back-EMF zero-crossing detection with the PWM. The A/D conversion of phase branch voltages is triggered at the centre of a PWM pulse. Then the voltage for Back-EMF is sensed at those time moments when the non-fed phase branch voltage is already stabilized.

The K60N512 is equipped with a Programmable Delay Block (PDB) synchronization of the FTM/PWM and ADC modules. Therefore, several assumptions have to be considered during the ADC sampling implementation. The first factor is the choice of PWM modulation. From this point of view, the complementary bipolar switching is the best choice. This PWM switching can be implemented in such a way that the duty cycle is in the range of 50–100 % at any time of motor operation. With unipolar switching, the duty cycle starts from 0 %. It means that there is limitation for ADC sampling at narrow pulses regardless of whether hardware synchronization is available or not. In this case, a different sensing method has to be chosen, where the Back-EMF voltage is compared to zero instead of the half DC bus voltage.

The right setting for the FTM/PWM module is shown in Figure 4-4. The FTM/PWM module is configured to run in complementary combined mode.

The PDB channel 1 can generate two triggers, Trigger0 and Trigger1, and one PDB interrupt. Trigger 0 is used for DC bus voltage, and Trigger 1 is used for BEMF voltage of phase A and phase B. The DC bus current measurement is done by a PDB interrupt using an ADC software trigger. After PDB0_IDLY is reached, the PDB calls interrupt service routine which switches a channel of the ADC to measure the DCB current and then calls the ADC software trigger. After conversion has completed and the result is read, the ADC channel is switched back to the BEMF measurement mode.

PDB channel 0 generates one trigger: Trigger 0, used for Back-EMF sensing of phase C.

This arrangement is appropriate because of the hardware interconnections. The result of the conversions can be read on the ADC0_isr or ADC1_isr interrupt. The sampling process can seen in Figure 4-4.

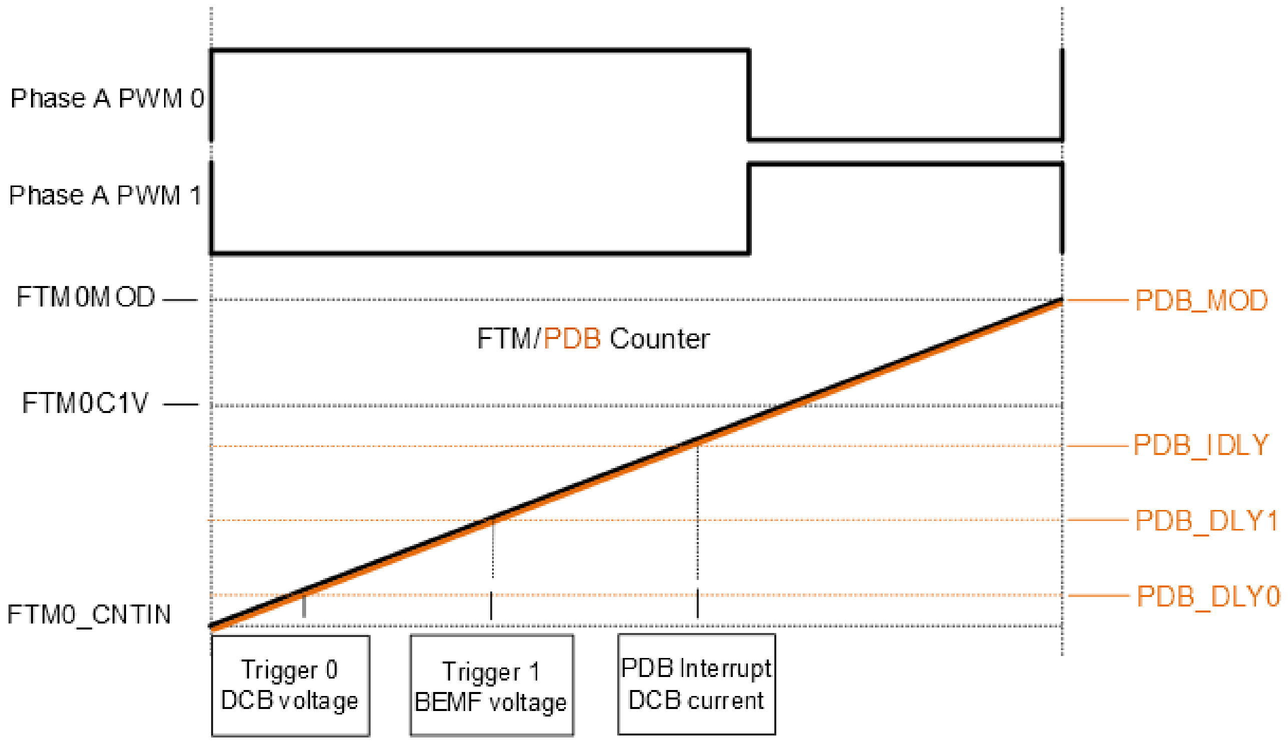


Figure 4-4. FTM/PWM to ADC Synchronization with PDB

4.4 Back-EMF zero-crossing sensing

The Back-EMF zero-crossing is detected by sensing the motor's non-fed phase—branch —voltage (u_{vi} in Back-EMF sensing) and the DC-bus voltage ud, using the ADC. To get the right Back-EMF voltage, two assumptions have to be made:

- Top and bottom switches (in diagonal) are driven by the same PWM signal
- No current is going through the non-fed phase used to sense the Back-EMF

The first assumption is achieved by the FTM/PWM module. The FTM/ADC synchronization at the centre of the PWM ON pulse is done by the PDB module. The second condition can be detected directly from the sensed Back-EMF voltage. As soon as the phase is disconnected from the DC bus, current still flows through the freewheeling diode. The conduction time depends on the momentary load of the motor. In some circumstances, the conduction time is so long that it does not allow for detection of Back-EMF voltage. The conduction freewheeling diode connects the released phase to either a positive or a negative DC bus voltage. The next step taken is done to detect current in the non-fed phase.

The first three samples after commutation are not considered for Back-EMF voltage detection because of transient event. The freewheeling delay can be changed in the reference design S/W in the SKIP_PWM_CYCLE constant in the BLDC.h file. The zero-crossing detection process can be seen in Figure 4-5.

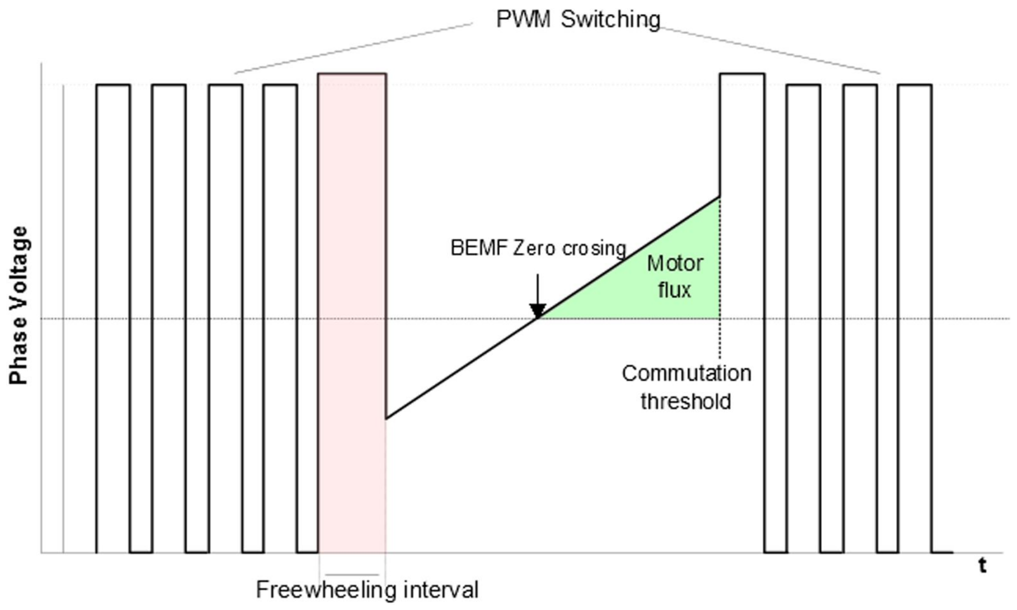


Figure 4-5. Zero-crossing detection process

4.5 Sensorless commutation control

This section presents a sensorless BLDC motor commutation with the Back-EMF zero-crossing technique and the Back-EMF integration technique. In order to start and run the BLDC motor, the control algorithm has to go through the following processes:

- Stop
- Alignment
- Run

First, the rotor is aligned to a known position without the positional feedback. When the rotor moves, the Back-EMF is induced on the non-fed phase and sensorless position detection can be employed. As a result, the position is known and can be used to calculate the speed and process the commutation in the Run state.

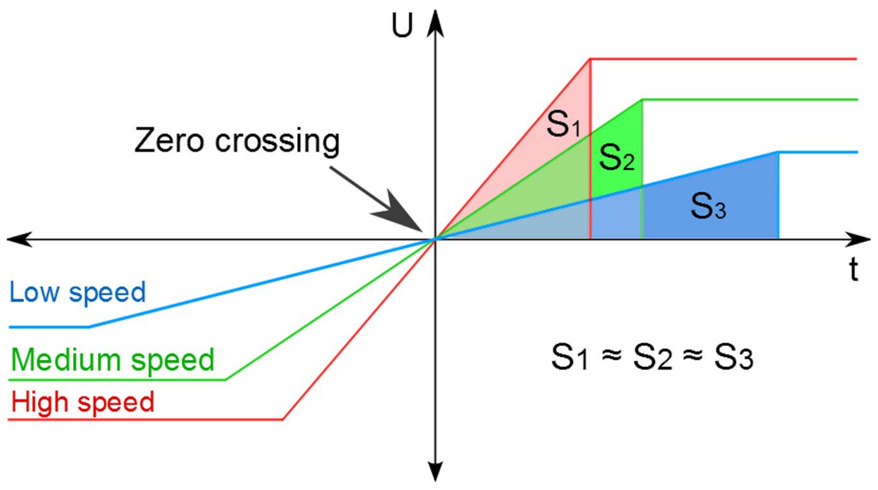


Figure 4-6. BEMF integration method

In this technique, the commutation instant is determined by integration of the non-fed phase’s Back-EMF, that is, the unexcited phase’s Back-EMF. The main characteristic is that the integrated area of the Back-EMFs shown in Figure 4-6 is approximately the same at all speeds (S1≈S2≈S3). The integration starts when the non-fed phase’s Back-EMF crosses zero. When the integrated value reaches a predefined threshold value, which corresponds to a commutation point, the phase current is commutated.

If a flux weakening operation is required, current advance can be achieved by changing the threshold voltage. The integration approach is less sensitive to switching noise and automatically adjusts for speed changes, but low-speed operation can be poor because of the error accumulation and offset voltage problems such as resistance precisions, noise, and so on.

In some ways, this technique has a better performance than the standard BEMF zero-crossing technique, but is more difficult for the CPU bandwidth. Therefore, this technique is subjected to a higher performance CPU, where the system application is not overloaded with other tasks. This technique can also be optimized for high speed using some multisampling methods.

4.6 Current (Torque) limitation

Besides the speed control, the drive has torque limitation. In this mode, the torque is limited to the desired value, which can be adjusted by the trimmer on the TWRMCLV3PH board. For overcurrent limit, the MC33937 overcurrent output pin is used. This output pin is periodically scanned in the PIT service routine. If an overcurrent is observed, the integral part of the speed controller is divided by three to limit the output duty cycle. If an overcurrent takes more than 400 ms, a fault has occurred. This value can be adjusted by OVER_CURRENT_TIME_THRESHOLD in the BLDC.h file. Overcurrent limitation is also performed by the MC33937 driver.

Chapter 5 Hardware

5.1 Hardware

The following hardware modules are needed for this application to function properly:

- TWR-K60N512
- TWR-Serial
- TWR-MC-LV3PH
- TWR-Elevator
- IAR J-Link LITE cortex
- BLDC Motor, LINIX 45ZWN24-40

For a proper functioning of the hardware, use the user manual: BLDCSLK60UG to configure the jumpers on all the hardware modules. This manual is available at http://www.freescale.com.

5.2 TWR-MC-LV3PH

The 3-phase Low-Voltage Motor Control board (TWR-MC-LV3PH) is a peripheral Tower System Module with one of the available MCU tower modules, accommodating a selected microcontroller, and provides a ready-made, software-development platform for one-third horsepower off-line motors. Feedback signals are provided that allow a variety of algorithms to control 3-phase PMSM and BLDC motors.

The TWR-MC-LV3PH module features:

- Power supply voltage input of 12-24 V DC, extended up to 50 V
- Output current up to 8 A
- Power supply reverse polarity protection circuitry
- 3-phase bridge inverter (6-MOSFETs)
- 3-phase MOSFET gate driver with overcurrent and undervoltage protection
- 3-phase and DC bus-current-sensing shunts
- DC bus-voltage sensing

- 3-phase Back-EMF voltage sensing circuitry
- Low-voltage on-board power supplies
- Encoder/Hall sensor sensing circuitry
- Motor power and signal connectors
- User LED, power-on LED, 6 PWM LED diodes

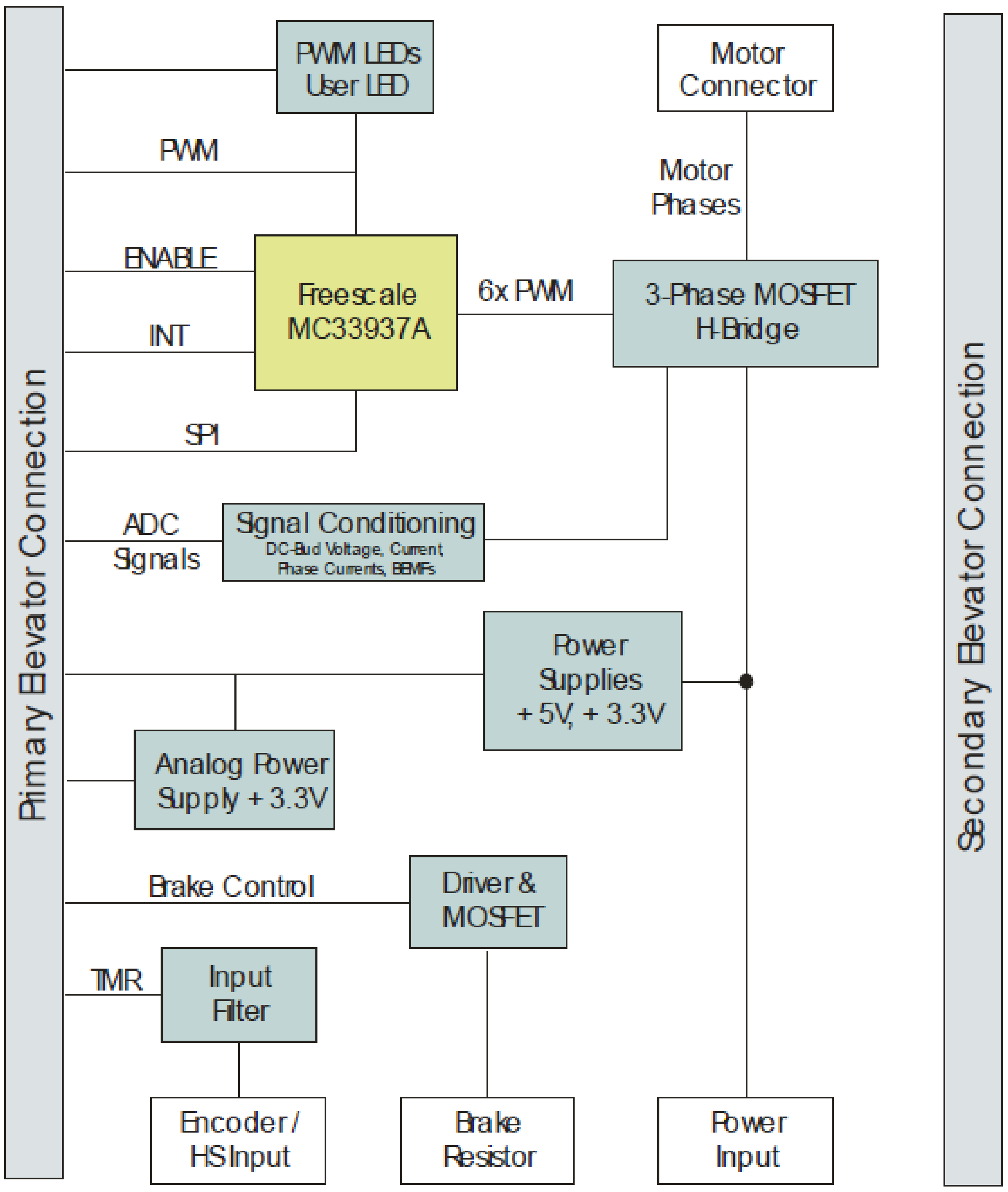


Figure 5-1. TWR-MC-LV3PH block diagram

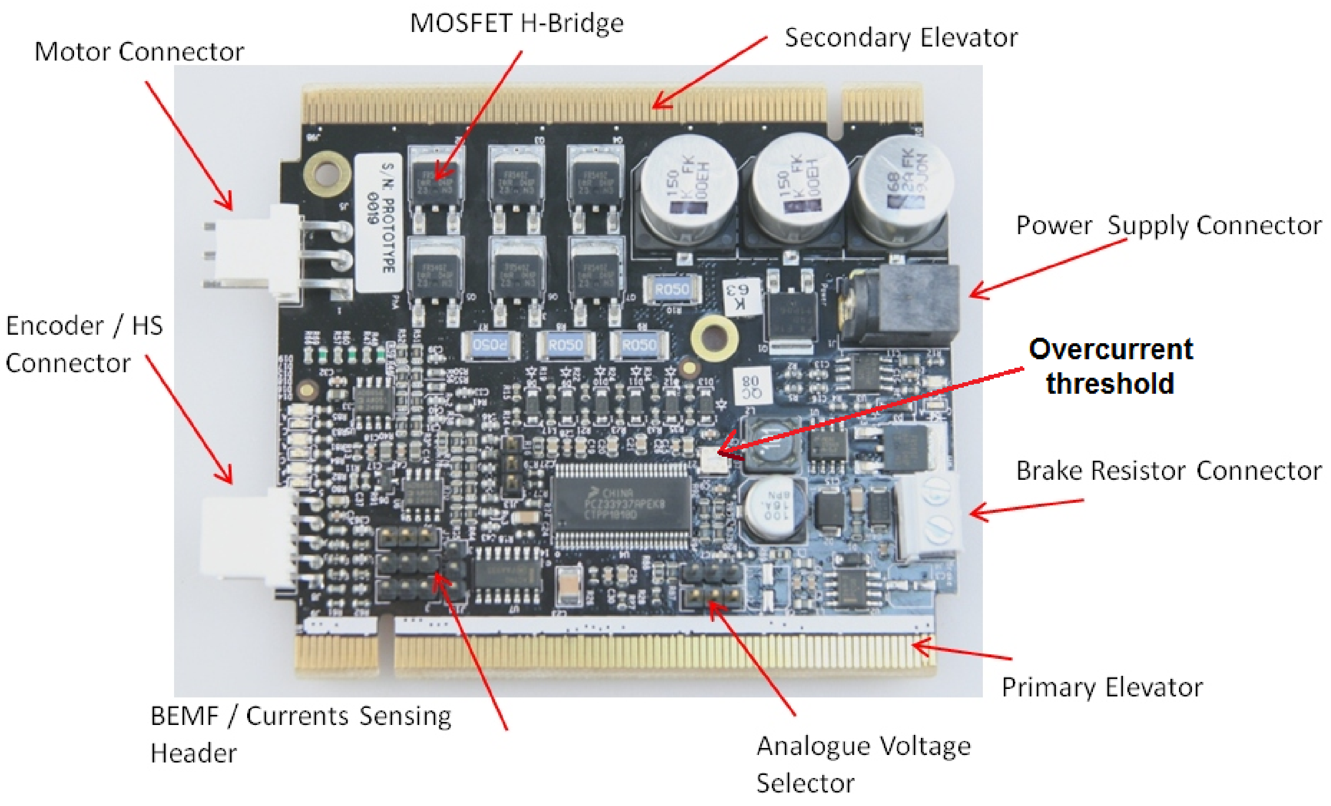


Figure 5-2. TWR-MC-LV3PH Image

5.3 TWR–K60N512

The TWR-K60N512 microcontroller module is part of the Freescale Tower System, a modular development platform that enables rapid prototyping and tool re-use through reconfigurable hardware.

- Kinetis K60N512 device (Cortex™-M4)
- Tower connectivity for access to USB, Ethernet, RS232/RS485, CAN, SPI, I²C,TWR-Serial
- Touch TWRPI socket adds support for various capacitive touch boards (key pads, rotary dials and sliders) TWR-MC-LV3PH
- Capacitive touch pads
- Integrated, open-source JTAG
- SD card slot
- MMA7660 3-axis accelerometer
- Tower plug-in (TWRPI) socket for expansion (sensors)
- flexbus
- Potentiometer, four LEDs, two pushbuttons, infrared port

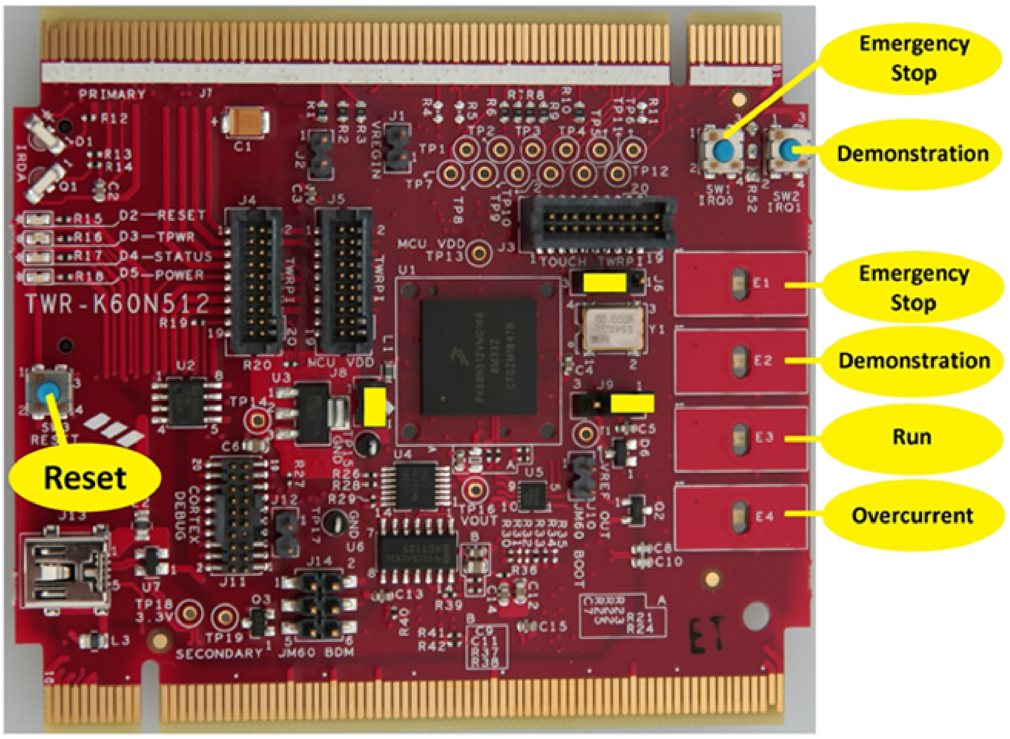


Figure 5-3. Image of TWR-K60N512

5.4 Motor LINIX 45ZWN24-40

The following motor is used by the BLDC Sensorless application. Of course, other motors can also be adapted to the application, just by defining and changing the motor related parameters. A detailed motor specification is shown below.

Table 5-1. Motor parameters

Characteristic	Symbol	Value	Units
Rated Voltage	Vt	24	V
Rated Speed @ Vt	—	4000	rpm
Rated torque	T	0.0924	Nm
Rated power	P	40	W
Continuous Current	Ics	2.34	A
Number of Pole Pairs	PP	2	—

Chapter 6
Software Design

6.1 Introduction

This section describes the design of the drive's software blocks. The software description comprises these topics:

- Application software main processes
- Processes flow charts
- FreeMASTER software
- Software setting, overview and application parameters

A basic data flow diagram and software interconnection are shown in Figure 6-1.

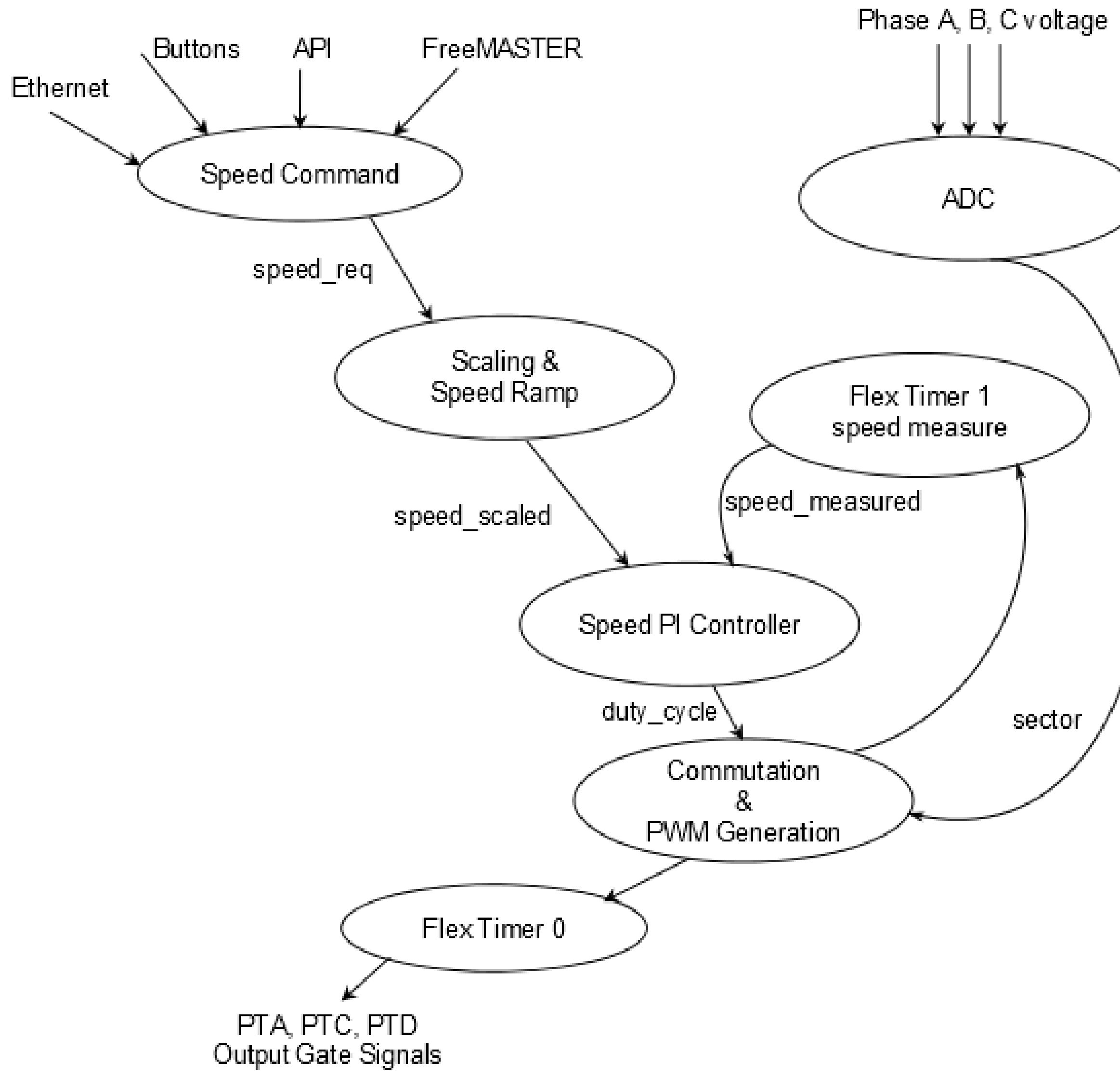


Figure 6-1. Data flow diagram

6.1.1 Application software main processes

The PK60N512VMD100 MCU runs the main control algorithm. According to the user interface and feedback signals, it generates 3-phase PWM output signals for a three-phase inverter.

The whole application runs on interrupts, for ease of use under MQX. In the main routine, there is only initialization of the microcontroller and an endless loop with a FreeMASTER poll function to control the application. When the required speed is other than zero, the application enables PWM output and starts a motor. The speed control loop is called periodically by a PIT 0. In this periodic loop, there is the major part of the software: a speed ramp, speed regulator, fault protection, application state machine, and so on.

ADC measurement is called by the ADC to PWM synchronization every 62.5 μs. After a conversion has completed, ADC0_isr or ADC1_isr is called.

The application data flow diagram with the main processes is shown in Figure 6-2 . The variables are described at the end of this section.

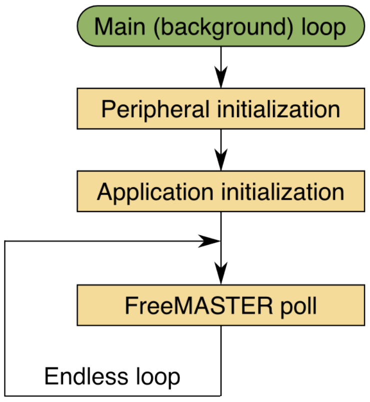


Figure 6-2. Main loop flow chart

As mentioned before, the complete motor control algorithm is driven by interrupts. The main function is used only for the MCU and application initialization; see Figure 6-2. When initialization terminates, the program goes into an endless loop. For all periodic functions, the periodic interrupt service routine is used. A flow chart of the periodic interrupt timer service routine is shown in Figure 6-3.

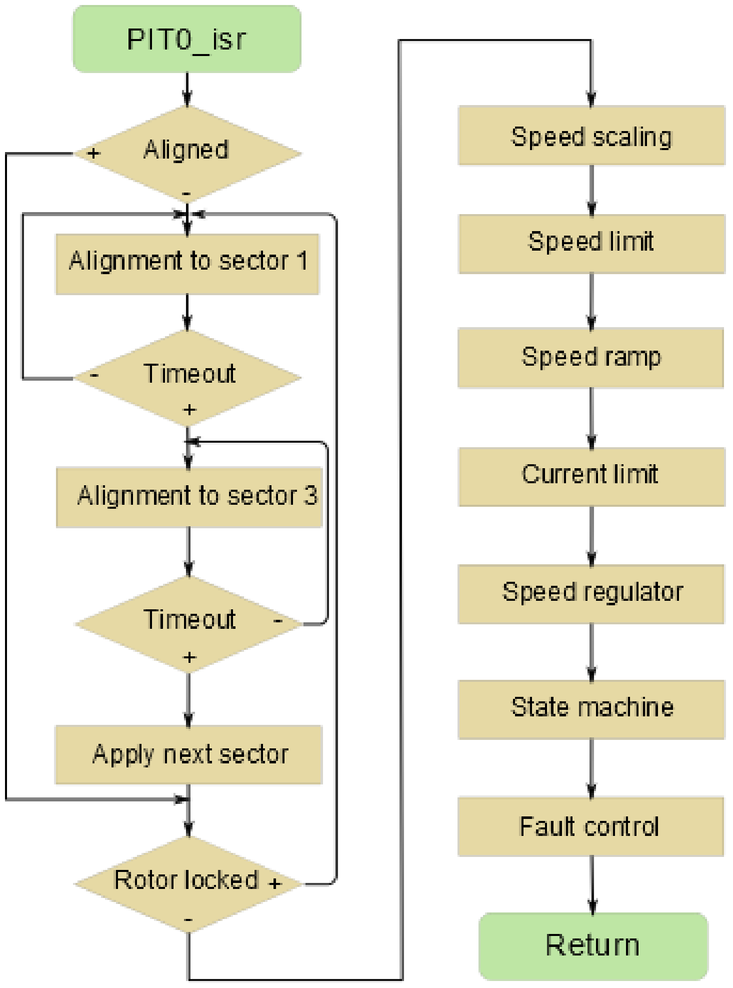


Figure 6-3. Periodic interrupt timer service routine flow chart

6.1.2 Process zero-crossing detection

This process is called every 62.5 ms in the ADC1_isr or ADC0_isr interrupt. The process is called after the A/D conversion, when a new sample of phase voltage/BEMF is available. This new sample is compared to the midpoint of the DC bus voltage. Details of zero-crossing detection are described in Back-EMF sensing . If zero-crossing is detected, the integration of the BEMF voltage is started. Every new sample is added to the integral_bemf after the half_dc_bus is subtracted. Then the new commutation event occurs after bemf_threshold is reached. ADC1_isr is more complex because ADC1 is used for DC bus current and DC bus voltage measurement. It is very important to recognize which result is in the result register. Flow charts of each service routine are shown in Figure 6-4 and Figure 6-5.

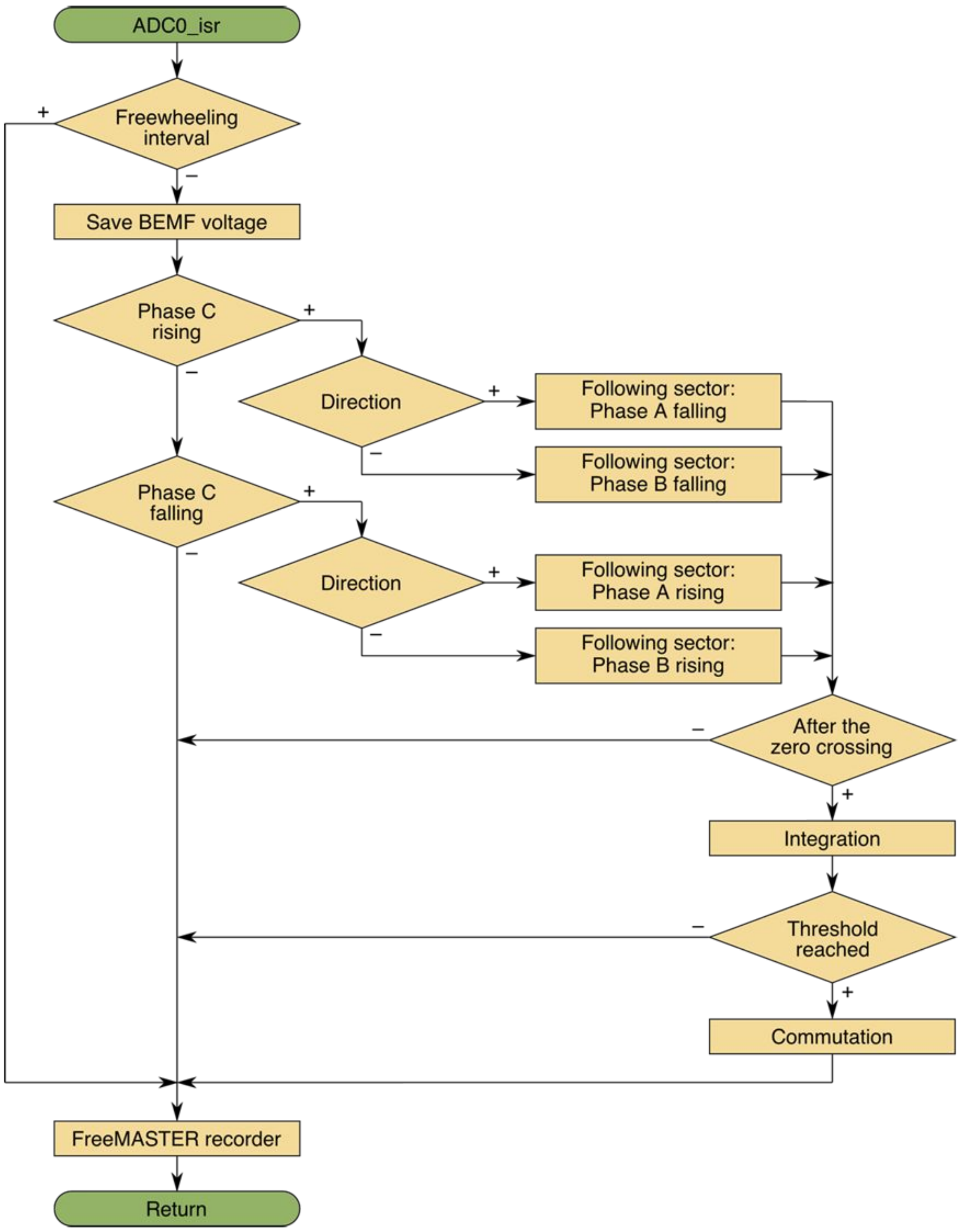


Figure 6-4. ADC0 interrupt service routine

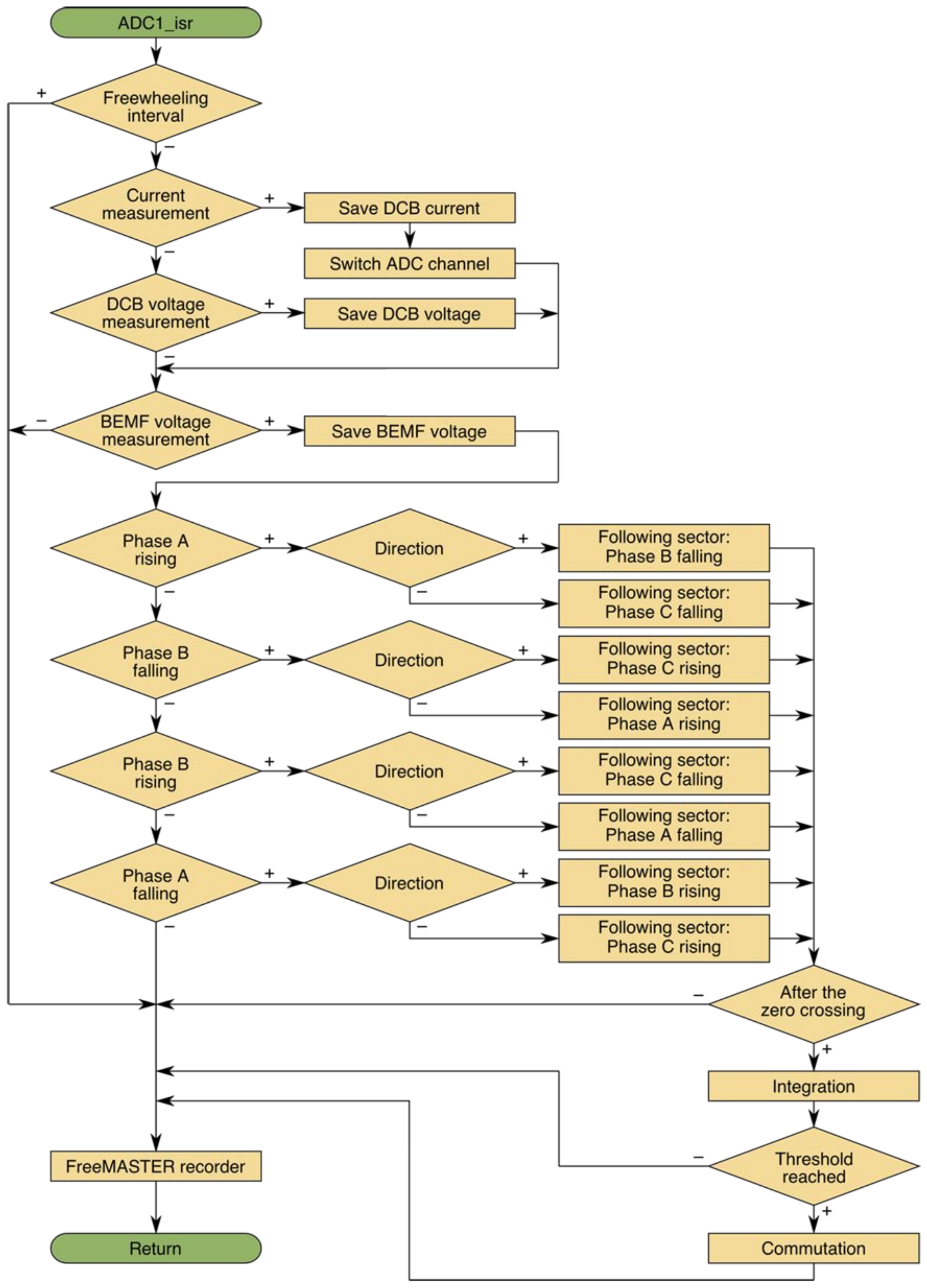


Figure 6-5. ADC1 interrupt service routine

6.1.3 Process speed calculation

The commutation period is filtered to get suitable input values to the speed PI controller. The value is calculated as the average from the last six commutation events. Then the averaged commutation period is recalculated as the motor speed. This process is executed every 1 ms in the *PIT0_isr* interrupt.

6.1.4 Process standstill detection

This process is used to measure time between two commutations. If the time between commutations is higher than defined, a standstill is detected. It also detects time of standstill duration. At any error, the motor is stopped and the start-up sequence is repeated. When the standstill detection is enabled, this process is executed every 1 ms in the *PIT0_isr* interrupt.

6.1.5 Process commutation

This process updates the variable *observed_sector*, updates the PWM period, and reconfigures the ADC and PDB modules for another phase measurement. The PWM update based on the new commutation sector is performed in the Mask_Swap process. The commutation process is called on every new commutation event. So, the execution period depends on the motor speed. Commutation is processed only if no fault has occurred. The flow chart of each service routine is shown in Figure 6-6.

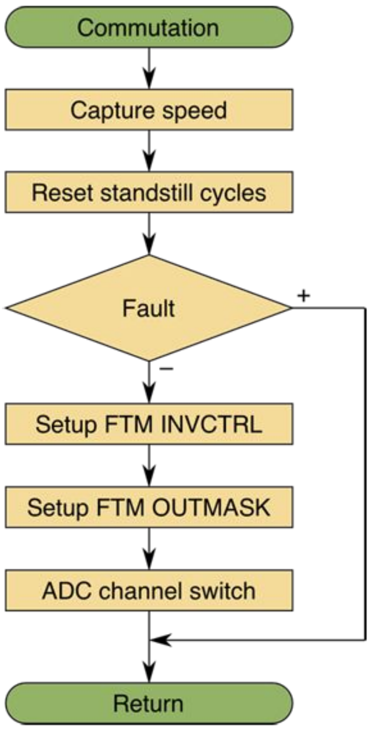


Figure 6-6. Commutation process

6.1.6 Process ramp

Since the overall application is a system with large inertia, it is necessary to profile the speed command when applying—otherwise it could overload the system possibilities. One approach is ramp generation. This ramp implements steps between the actual speed and the speed command. The process is executed every 1 ms in the *PIT0_isr* interrupt. In this application, a ramp is used from the library GFLIB. This ramp needs three parameters.

The first two parameters are the up and down angles of the ramp. The last parameter is the required speed. All three parameters are in 32-bit fractional data format. It is possible to enter the first two parameters in *BLDC_config.h* as integers in rpm/s, but they are recalculated back to frac32 during application initialization. The parameters are named *SPEED_RAMP_UP* and *SPEED_RAMP_DOWN*.

The ramp execution is highly dependent on the PIT period. The default value is 1 ms. If the period of PIT is changed, it is necessary to change the PIT period in the RAMP_SCALE_CONST macro. This macro is used to compute *SPEED_RAMP_UP* and *SPEED_RAMP_DOWN* from int to frac32. The second possibility for entering the speed ramp parameters is to use FreeMASTER. In FreeMASTER, data can be entered as an integer in rpm/s at program runtime. In the MQX version, ramp parameters can be changed using web server buttons.

6.1.7 Process FreeMASTER

The FreeMASTER process is part of the application software. FreeMASTER communication is implemented fully by the SCI interface. FreeMASTER writes new values through the SCI interface into data RAM, where particular variables are located. This operation is executed every time the FreeMASTER *FMSTR_Poll* and *FMSTR_Recorder* functions are executed.

6.1.8 Process PWM update

This process generates the correct voltage pattern on the motor, based on the actual PWM sector, and required output voltage. The process is called every 1 ms in the PIT0_isr interrupt . The commutation table is derived from Back-EMF voltage measured on the motor, see Figure 6-7. Besides the proper commutation pattern, the Back-EMF sensing can be evaluated from Figure 6-7. For example, if phase C is connected to a positive DC bus voltage and phase B is connected to a negative DC bus voltage, the falling Back-EMF voltage of phase A has to be evaluated for zero-crossing. The resultant commutation table includes sensing of the Back-EMF voltage, as can be seen in Table 6-1. Commutation table for counter-clockwise direction.

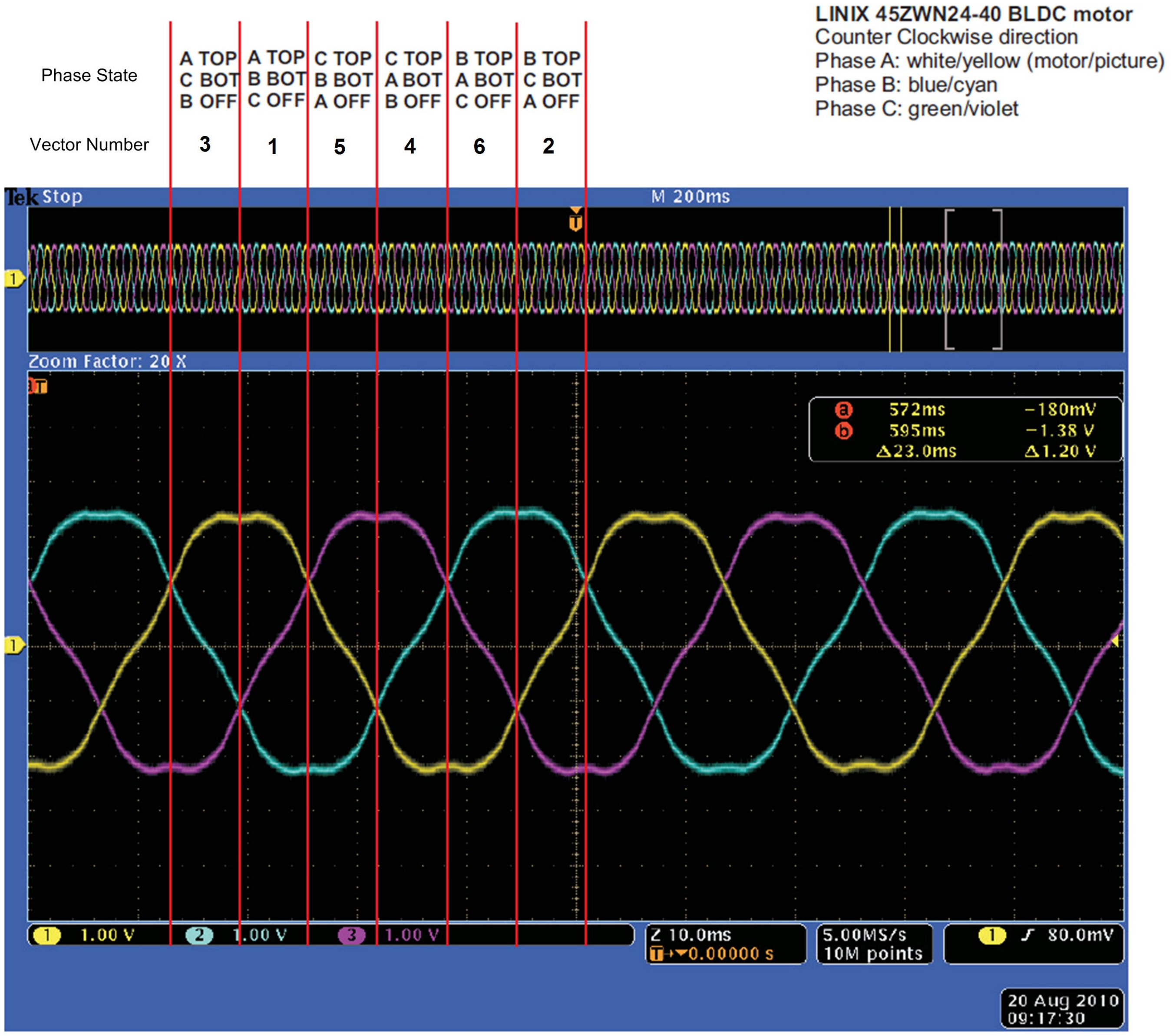


Figure 6-7. Motor BEMF voltage

Table 6-1. Commutation table

Vector number		Phase		Voltage sensing
	A	B	C	
1	DCB+	DCB-	NC	PHASE_C_RISING
5	NC	DCB-	DCB+	PHASE_A_FALLING
4	DCB-	NC	DCB+	PHASE_B_RISING
6	DCB-	DCB+	NC	PHASE_C_FALLING
2	NC	DCB+	DCB-	PHASE_A_RISING
3	DCB+	NC	DCB-	PHASE_B_FALLING

The application constants are split into **BLDC.h** and **BLDCconfig.h**. File **BLDCconfig.h** deals with main motor parameters such as: electrical and mechanical parameters of the BLDC, the speed scale, and parameters of particular states like alignment, start-up, and run. File **BLDC.h** deals with the application parameters, which are needed only for a detailed setting of the application.

6.1.9 Process sequence error and current measurement

PDB_error_isr is used for two purposes. The first function of this interrupt is to clear an error flag which can occur because of a sequence error. A sequence error typically happens because a delay is set too short and a pre-trigger asserts before the previously triggered ADC conversion has completed. For example, a sequence error occur when a breakpoint is included into the program. It is necessary to disable the PBD block, clear the error flag, and enable the PDB block again.

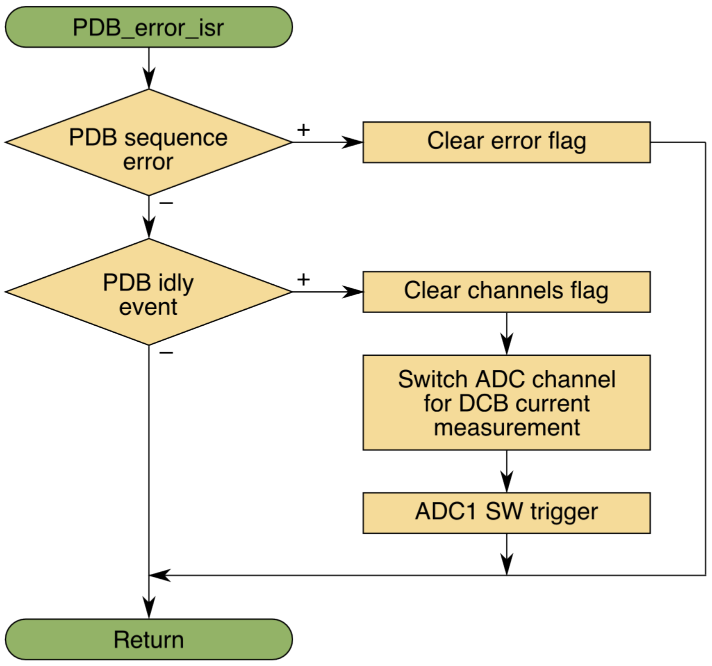


Figure 6-8. PDB interrupt service routine flow chart

The second purpose of this interrupt is for current measurement. When the *PDB_IDLY* is reached, the *PDB_error_isr* interrupt is called. In this process, the ADC is reconfigured to Software Trigger mode. The measurement is started on ADC1 channel 12, where a DCB current signal is connected. When the conversion has completed, the result of the DCB current is saved and the ADC1 is reconfigured back to DCB voltage measurement. Flow charts of this process are shown in Figure 6-8 and Figure 6-5.

6.2 Process manual control

Software can be controlled by three buttons that are periodically scanned in the *PIT_isr* loop. All buttons are placed on the TWR-K60N512 board. Button SW1 is used for an emergency stop of the motor. Button SW2 is used for demonstration of the application. The last button is SW3, which is used for a processor reset. There is also a potentiometer on the TWR-MC-LV3PH board used for the overcurrent level setting.

6.3 Process speed PI controller

The speed PI control algorithm processes the *speed_error* between *speed_scaled* and *speed_measured*. The PI controller output is passed to the PWM generator as a newly corrected value of the applied motor voltage.

The PI controller routine is calculated in the interrupt routine of the PIT device, in PIT0_isr, which is called every 1 ms. The integral part of the PI controller is disabled at low speeds, under 299 RPM, because in that case measurement of speed is not accurate and the PI controller can be unstable. In the program there are two macros: MIN_CW_SPEED_32 and MIN_CWW_SPEED_32 to determine when the integral part of PI should be disabled.

The first input value of the PI controller is the variable *speed_scaled* that is the output from the ramp algorithm. The second input is the actual *speed_measured*. The other two inputs are pointers to the structure of the PI controller parameters *trMyPI*. All these parameters are used by the PI controller function GFLIB_ControllerPIp.

The output of this function is *s32Output*. It is scaled to the PWM scale as *delta_duty* and is added to *half_duty*. The result of this process is *duty_cycle* which is loaded into the Flex Timer registers.

The PI speed controller parameters must be configured each time the speed scale is changed or when the motor is changed.

6.4 Interrupt installation

The difference between the bare-metal version and the MQX version is only in the method used to install interrupts. The method used depends on whether MQX or Bare_Metal is predefined in the IAR project options (Options / C/C++Compiler / Preprocessor / Defined symbols).

In the bare-metal version, we can install an interrupt directly. You can simply allow an interrupt by setting the right bit in the NVICISER register. Priority of the interrupt can be configured using the register NVIC_IP. For a better understanding, see the following example.

Installation of interrupts:

```
NVICICPR1 = (1 << (25));  // clear a possible pending interrupt first
NVICISER1 = (1 << (25));  // enable ADC0 interrupt

NVICICPR1 = (1 << (26));  // clear a possible pending interrupt first
NVICISER1 = (1 << (26));  // enable ADC1 interrupt

NVICICPR2 = (1 << (4));   // clear a possible pending interrupt first
NVICISER2 = (1 << (4));   // enable PIT0 interrupt

NVICICPR2 = (1 << (8));   // clear a possible pending interrupt first
NVICISER2 = (1 << (8));   // enable PDB interrupt
```

Setup priority of interrupts:

```
NVIC_IP(68) = 0xC0;        // set priority for PIT0
NVIC_IP(58) = 0x90;        // set priority for ADC1
NVIC_IP(57) = 0xA0;        // set priority for ADC0
NVIC_IP(72) = 0x80;        // set priority for PDB
```

If the installation of interrupts in MQX is the same as in the bare-metal version, MQX would not work correctly. The MQX's own interrupts are quite slow for motor control applications. The best solution for high-speed applications is kernel interrupts. The kernel interrupts are natural CPU interrupts with no MQX overhead and minimal execution duration. The disadvantage of the kernel interrupt is that no MQX functionalities, such as events or semaphores, are supported. For a better understanding of how to use kernel interrupts, see the following example.

Installation of interrupts:

```
 _int_install_kernel_isr(INT_PIT0,PIT0_isr);
 _int_install_kernel_isr(INT_ADC0,ADC0_isr);
 _int_install_kernel_isr(INT_ADC1,ADC1_isr);
 _int_install_kernel_isr(INT_PDB0,PDB_error_isr);
```

Set priority of interrupts:

```
_bsp_int_init((IRQInterruptIndex)INT_PIT0, 1, 0, 1);
_bsp_int_init((IRQInterruptIndex)INT_PDB0, 2, 0, 1);
_bsp_int_init((IRQInterruptIndex)INT_ADC0, 0, 0, 1);
_bsp_int_init((IRQInterruptIndex)INT_ADC1, 0, 0, 1);
```

6.5 BLDC motor parameters (BLDCconfig.h)

6.5.1 Speed scale parameters

The following parameters are used for the “*speed_ measured*” scaling. An incorrect setting of these parameters will also cause an incorrect functioning of the speed regulator.

```
#define PP                      2         - Number of motor pole pairs [1…8]
#define TPM_C                  48e6       - Flex Timer 1 input clock
#define TPM_P                   128       - Prescaler of Flex Timer 1 [1…128]
#define MAX_SCALED_SPEED       5000.0     - Maximal theoretic speed (include reserve)
                                          - Decimal point is necessary.
```

6.5.2 Applications parameters

```
#define USE_FREEMASTER          1       -FreeMASTER usage in application [0/1]
#define PWM_FREQ             16000       - PWM frequency [5000 … 20000]
#define BEMF_THRESHOLD        2000       - Commutation threshold
#define START_DUTY_CYCLE        25.0     - Duty cycle during alignment [0…100%]
                                        - Decimal point is NECESSARY!
```

Ramp settings:

```
#define SPEED_RAMP_UP         4000      - Speed up ramp coefficient [100 …20000]
#define SPEED_RAMP_DOWN       4000      - Speed down ramp coefficient [100 …20000]
```

PI controller setting

```
#define PI_PROP_GAIN           0.33          - P parameter of PI
#define PI_INTEG_GAIN          0.0035        - I parameter of PI
#define PI_PROP_GAIN_SHIFT     1             - Proportional Gain Shift
#define PI_INTEG_GAIN_SHIFT    1             - Integral Gain Shift
#define PI_INTEG_PART          0             - Integral part of PI
```

6.6 Applications parameters (BLDC.h)

```
  #define FULL_DUTY ( TPM_C / PWM_FREQ )  -100% duty cycle
  #define HALF_DUTY FULL_DUTY/2           - 50% duty cycle
```

This constant defines the value of the minimal required speed (default value: ± 400 rpm)

```
  #define MIN_CW_SPEED FRAC16(400.0/MAX_SCALED_SPEED)
  #define MIN_CWW_SPEED FRAC16(-400.0/MAX_SCALED_SPEED)
```

This constant defines the value of the maximal required speed (default value: 80% of the MAX_SCALED_SPEED)

```
  #define MAX_CW_SPEED FRAC16(0.8)
  #define MAX_CWW_SPEED FRAC16(-0.8)
```

This constant defines the value of the minimal measurable speed (default value: ±200 rpm)

```
  #define MIN_MES_CW_SPEED FRAC16 (200.0/MAX_SCALED_SPEED)
  #define MIN_MES_CWW_SPEED FRAC16 (-200.0/MAX_SCALED_SPEED)
```

This constant defines when the integral part of regulator is disabled for better stability (default value: ±299)

```
#define MIN_CW_SPEED_32 299.0
#define MIN_CWW_SPEED_32 -299.0
```

This constant is used to scale fraction parameters of the speed ramp to real physical units [rpm/s]. It depends on the frequency of the PIT. Number 0.00104 means a period PIT (1.04 ms)

```
#define RAMP_SCALE_CONST ( 0.00104 / MAX_SCALED_SPEED )
```

This constant is used to scale fraction parameters of the speed to real physical units [rpm]. Number 32768 is the maximum value of a signed integer

```
#define SPEED_TO_RPM_SCALE ( 32768.0 / MAX_SCALED_SPEED )
```

This is the speed scale constant. For more information, see Speed scale ,

```
#define SCALE_CONST (unsigned long) ( 6 * ( 60 / (PP*(TPM_P/TPM_C))) / MAX_SCALED_SPEED )
```

This constant defines how many PWM cycles are skipped during a freewheel diode interval before a BEMF measurement starts. To that value is always added the number 1. Thus, the minimal possible output value is 1, when a zero is entered into this constant (Default value: 2)

```
#define SKIP_PWM_CYCLE 2
```

This constant defines the maximal time between two commutations for the rotor standstill detection. Depends on the PIT timer frequency. (Default value: 25 ms)

```
#define STAND_THRESHOLD 25
```

This constant defines the time to make an alignment. Depends on the PIT timer frequency (Default value: 6)

```
#define ALIGNMENT_CYCLE 6
```

In this application, an alignment in two sectors is performed. This constant defines the time when the alignment is switched to the second sector

```
#define ALIGNMENT_CYCLE_HALF ALIGNMENT_CYCLE /2
```

This constant defines the minimal DC bus voltage. Constant is not scaled, so it is a relative value (Default value:170 is adequate to 12 V)

```
#define MIN_DC_BUS 170 // Minimal DC BUS voltage ( 12 V)
```

This constant defines the minimal DC bus voltage. Constant is not scaled, so it is a relative value (Default value:410 is adequate to 29 V)

```
#define MAX_DC_BUS 410 // Maximal DC BUS voltage ( 29 V)
```

This constant defines the time period when an overcurrent fault is triggered. Depends on the PIT timer frequency. (Default value:400 ms)

```
#define OVER_CURRENT_TIME_THRESHOLD 400
```

6.7 Speed scale

This application uses a fractional representation for speed. The N-bit signed fractional format is represented using the1. [N-1] format (1 sign bit, N-1 fractional bits). Signed fractional numbers (SF) lie in the following range:

$$-1.0 \leq \mathrm{SF} \leq +1.0 - 2^{-[N-1]}$$

For words and long-word signed fractions, the most negative number that can be represented is –1.0, whose internal representation is \$8000 and \$80000000, respectively. The most positive word is \$7FFF or 1.0 – 2-15, and the most positive long-word is \$7FFFFFFF or 1.0 – 2-31. The following equation shows the relationship between a real and a fractional representation:

$$Fractional\,Value = \frac{Real\,Value}{Real\,Quatity\,Range}$$

Fractional Value = Fractional representation of speed quantities [–]

Real Value = Real speed quantities in physical units [rpm]

Real Quantity Range = Max. defined speed used for scaling in physical units [rpm]

In this application the scale constant is calculated by this equation:

$$SCALE\_CONST = \frac{6}{MAX\_SCALED\_SPEED} x \frac{60}{PP\frac{TPM\_P}{TPM\_C}} = \frac{6}{5000} x \frac{60}{2\frac{128}{48000000}} = 13500$$

The following macro is used to compute the speed scale:

SCALE_CONST (tFrac32)(6*(60 / (PP*(TPM_P/TPM_C))) / MAX_SCALED_SPEED)

PP — Number of pole pairs of the motor used. Default value is 2.

TPM_C — Input clock of the timer in Hz. Default value is 48e6.

TPM_P — Prescaler of the timer input clock. Default value is 128.

MAX_SCALED_SPEED — Maximal speed with reserve. Default value is 5000.

As you can see from the previous equation, this scale constant is multiplied by 6. This is done for the average compute. The *time_measured* variable is used to store the sum of 6 time intervals.

```
for(int g=1 ; g < 7 ; g++)
  { time_measured = time_measured + period[g]; }
```

Then, there is no need to divide the sum of the time intervals by 6, because, in SCALE_CONST, this is already included.

Finally, the *speed_measured* is calculated by this equation:

$$\text{measured speed} = \frac{1}{\text{time measured}} * \text{SCALE CONST}$$

Chapter 7
Sensorless BLDC Demo Operation

7.1 Application control

There are two versions of the application software:

- BLDC under the MQX RTOS and web server
- BLDC on bare-metal

Both applications can be also controlled by the FreeMASTER software. This FreeMASTER PC application allows real-time monitoring or modification of all required variables through an easy and user-friendly graphical user interface. Selected variables can be also monitored in a time domain scope representation. The MQX version can be controlled from any web browser on a PC via the Ethernet communication interface, because in this version the web server is implemented. Both versions of the software can be controlled by three buttons on the TWR-K60N512 board. Button SW1 is used for an emergency stop of the motor. Button SW2 is used for demonstration of the application. The last button is SW3, which is used for a processor reset. There is also a potentiometer the TWR-MC-LV3PH board used for the overcurrent level setting.

7.2 BLDC drive operational modes

7.2.1 Speed-closed loop

If the application is set to the speed-closed loop mode, the user can set the required speed by the FreeMASTER or web server. The DC bus current (torque) is automatically limited according the potentiometer on the TWR-MC-LV3PH board. In this case, the BLDC drive maintains the required speed until the maximal DC bus current (torque) is exceeded.

7.2.2 Speed-open loop

Enables or disables the speed regulator. To switch to this mode, it is necessary to change the parameter “Closed Loop” to “disable” in the FreeMASTER control interface. Switching to this mode is possible only when the required speed is other than zero, because the start-up sequence depends on the required speed variations. In this mode, the duty cycle parameter can be changed from 50% to 100%.

7.2.3 Standstill detection

If a rotor is locked, the software periodically tries to restart it. If the required speed is equal to zero, the software turns off the PWM, if a motor is stopped for more than 2 s. To switch to this mode, it is necessary to change the parameter “Standstill Detection” to “disable” in the FreeMASTER control interface.

7.3 Control interfaces

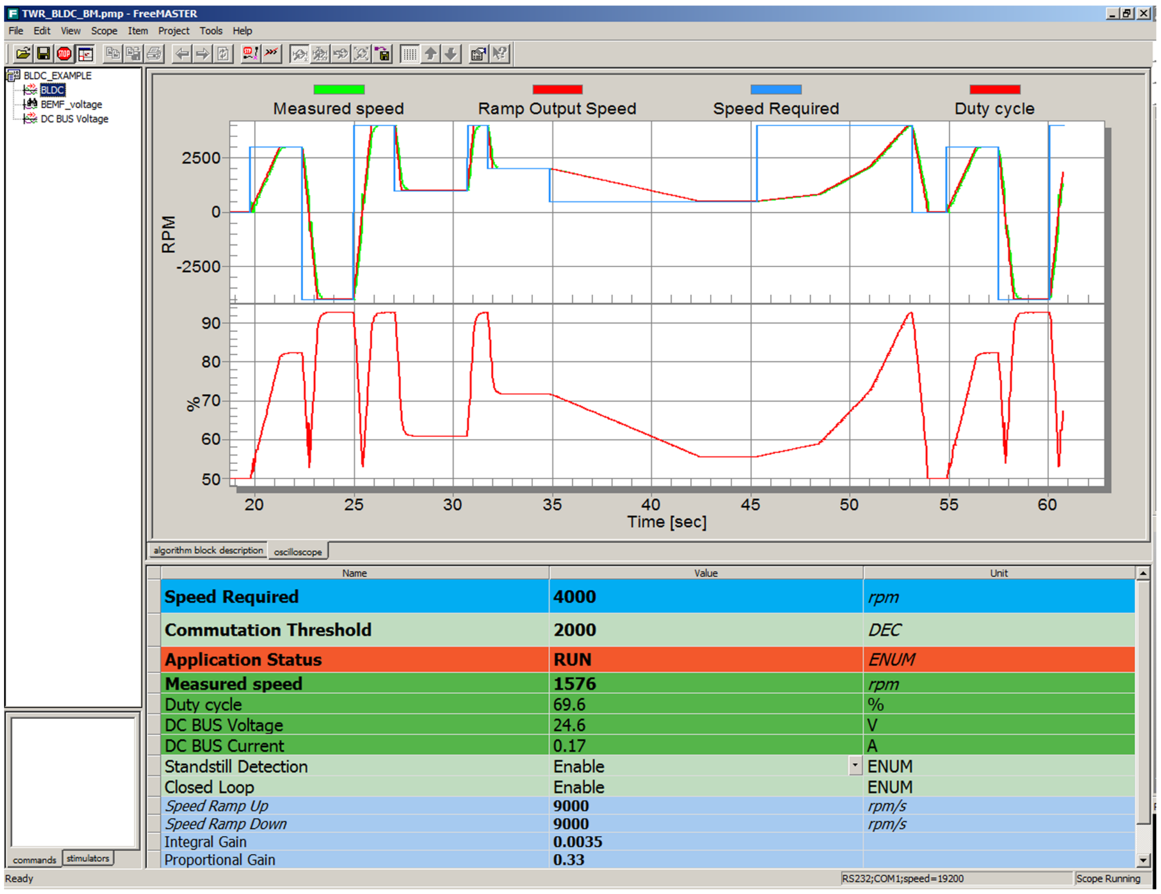
7.3.1 FreeMASTER control

Remote operation can be provided by FreeMASTER software via the SCI to USB interface. For a correct FreeMASTER operation, follow the given steps:

1. Open FreeMASTER and go to Project/Options/Comm and set communication via to the Direct RS232.
2. Select the COM port where the TWR-SERIAL is connected (see System Properties/ Device Manager/Ports). The communication speed is 19200 Bd.
3. The next step is to toggle the communication button. After that, in the bottom right-hand corner, there should be RS232;COMxx;19200, which means that communication has been established.
4. If not, toggle (STOP) the communication and unplug/plug the FreeMASTER USB cable. Then toggle (START) the communication button.

Detailed information about the set-up can be found in the user guide: BLDCSLK60UG, available at http://www.freescale.com.

After launching the application and performing all necessary settings, click the scope: “BLDC” item in the project tree structure of the FreeMASTER application window, as shown in Figure 7-1 . In this view, variables used for the application state, speed, PI controller, and ramp settings are visible. For the demonstration purposes, this is sufficient.



Name	Value	Unit
Speed Required	4000	rpm
Commutation Threshold	2000	DEC
Application Status	RUN	ENUM
Measured speed	1576	rpm
Duty cycle	69.6	%
DC BUS Voltage	24.6	V
DC BUS Current	0.17	A
Standstill Detection	Enable	ENUM
Closed Loop	Enable	ENUM
Speed Ramp Up	9000	*rpm/s*
Speed Ramp Down	9000	*rpm/s*
Integral Gain	0.0035	
Proportional Gain	0.33	

Figure 7-1. FreeMASTER control interface

Description of variables :

Speed Required : This variable serves for entering the required speed of the motor and the direction of motion. If the number is negative, the motor runs in a counter-clockwise direction, and when it is positive the motor runs clockwise. You can modify this variable from -400 to -4000, and from 400 to 4000. Any other numbers will be ignored.

Commutation Threshold : For commutation timing. (BEMF integral threshold)

Measured Speed : Variable shows the actual motor speed.

Standstill Detection : Detection of a rotor standstill. If a rotor is locked, the software periodically tries to restart it. If the required speed is equal to zero the software turns off the PWM, if a motor is stopped for more than 2 s.

Closed-Loop : Enable or disable the speed regulator. If it is disabled, the “Duty cycle” parameter can be changed. [from 50% to 100%].

Integral Gain, Proportional Gain : Speed PI controller setting.

trMyRamp.s32RampDown : Motor acceleration. Can be set in rpm/s.

trMyRamp.s32RampUp : Motor deceleration. Can be set in rpm/s.

7.3.2 Web server control

The MQX version of the application can be controlled through a web server. The following setting must be done for a proper web server function:

1. Disconnect the computer from any other network (both wired and wireless)
2. Enable automatic IP configuration on your computer.
3. Turn off all proxy servers.
4. Connect the demo to the computer through an Ethernet cable.
5. Restart the Ethernet adapter.
6. Wait to establish a connection.
7. Write the IP address of the demo device into Internet Explorer. The default IP address is 169.254.3.3. And then press enter.

NOTE

The web server is optimized only for MS Internet Explorer 8.

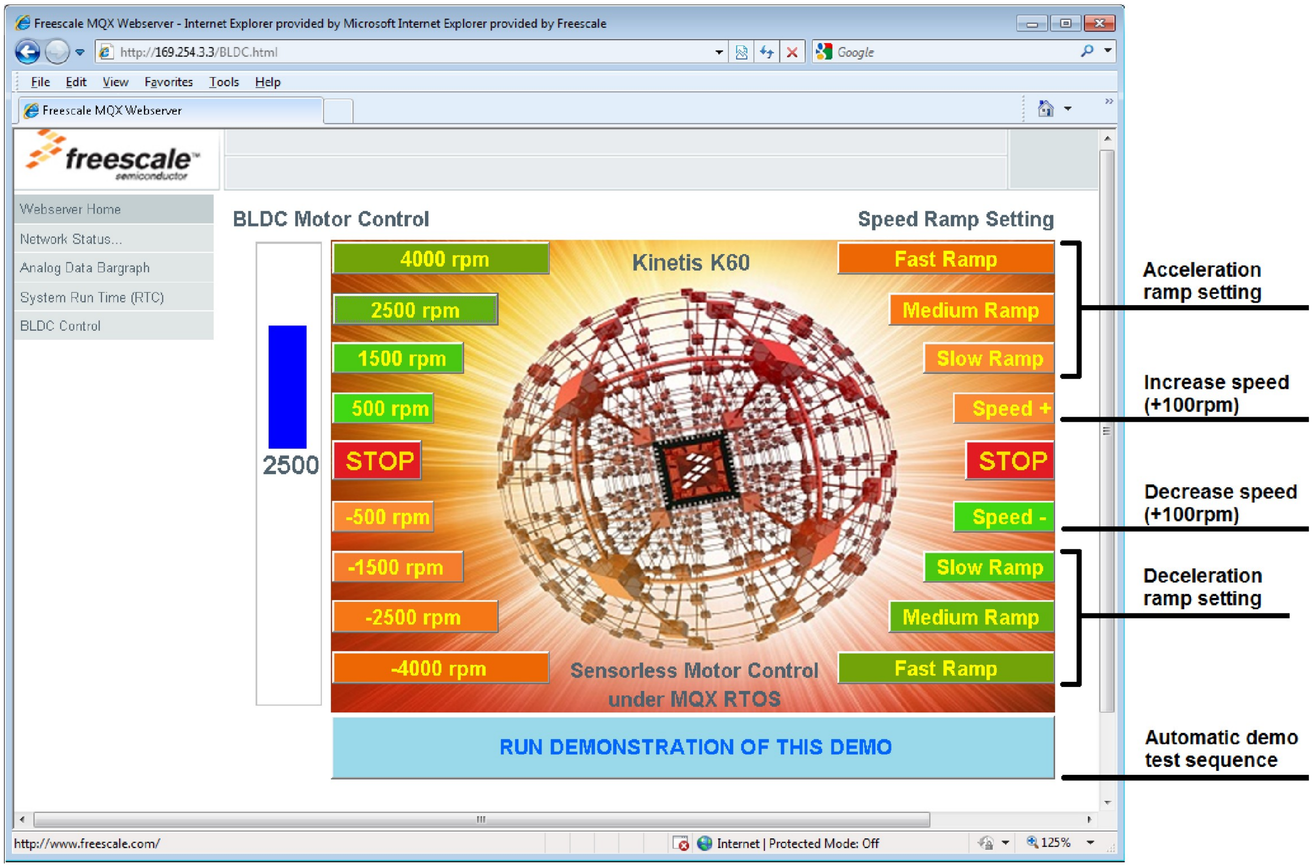


Figure 7-2. Web page for application control

For more information about the web server setting and application configuration, see BLDCSLK60UG which is available at http://www.freescale.com.

7.4 Integration of the motor control driver into other applications

Both versions of the motor control driver can be part of other applications. The API is defined by the following six functions.

1. **void** Set_speed(**int** motor_number, **signed short** speed_input)

   First parameter is the number of the motor which receives the command. This demo has only one motor, so enter 1.

   Second parameter is the input speed in **signed short** format. The input value is in rpm.

   Return : void

2. **unsigned char** Get_status(**void**)

   Return: Status of the application

0-IDLE, 1-STOP, 2-RUNNING, 3-ALIGNMENT, 6-EMERGENCY_STOP, 7-UNDER_VOLTAGE_FAULT , 8-OVER_VOLTAGE_FAULT, 9-OVER_CURRENT_FAULT

3. **signed short** Get_speed(**int** motor_number)

   First parameter is the number of the motor which receives the command. This demo has only one motor, so enter 1.

   Return: Measured speed in **signed short** data format. The value is in rpm.

4. **signed short** Get_req_speed(**int** motor_number)

   First parameter is the number of the motor which receives the command.

   Return: Required speed in **signed short** data format. The value is in rpm .

5. **void** Set_ramp_up(int motor_number, **int** ramp_up)

   First parameter is the number of the motor which receives the command.

   Second parameter : Ramp up profile . The value is in rpm per second.

   Return: void

6. **void** Set_ramp_down(int motor_number, **int** ramp_down)

   First parameter is the number of the motor which receives the command.

   Second parameter : Ramp down Profile. The value is in rpm per second.

   Return: void

Chapter 8
Software And Peripherals Overview

8.1 Software listing

The code is written in C (IAR Embedded Workbench for ARM 6.3). The software consists of the following code files:

- application source code files:
  - main.c
  - BLDC.C
  - Demonstration.c
  - peripheral.c
  - MC33927.c
- application header files:
  - main.h
  - BLDC.h
  - BLDC_config.h
  - peripheral.h
  - MC33927.h
  - freemaster_cfg.h

The application source file **main.c** contains the following software routines:

- *main()* - This is the entry point following a Reset. It calls the initialization routines.
- *FreeMasterInit()* – Initialization of FreeMASTER.

The application source file **BLDC.c** includes the following software routines:

- *SPI_Send()* - For communication with the MC33937.
- *MCU_init()* - Initialization routine for peripherals and interrupts.
- *Clock_init()* - Initialization of the peripherals clock.
- *Variables_reset()* - Initialization of variables after a reset.
- *App_state_machine()*- State machine execution.
- *Commutation()* - Reconfigure and update the FlexTimer registers for PWM generation.

- *PIT0_isr()* - Periodic interrupt timer service routine (1 ms). Executes the major part of the motor control algorithm. (Speed controller, fault protection, ramp execution, speed scaling, standstill detection, and so on)
- *ADC0_isr()* - ADC0 conversion complete.
- *ADC1_isr()* - ADC1 conversion complete.
- *PDB_error_isr()* - Service for the PDB error and for the ADC channel switch for current measurement.
- *Threshold_rising_bemf()* - Observing the BEMF voltage up slope. Searching for a zero-crossing, integrating the BEMF voltage and performing commutation after the threshold is reached.
- *Threshold_falling_bemf()* - Observing the BEMF voltage down slope. Searching for zero-crossing, integrating the BEMF voltage and performing commutation after the threshold is reached.
- *Speed_measure( )* - Save the FlexTimer 1 output value for computing the speed..
- *Emergency_Stop()* - Service of hazardous states.
- *Set_duty_cycle()* - Update the duty cycle of the PWM
- *Mask_Swap()* - Reconfigure PWM output channel according to commutation table.
- *Demonstration()* - Test sequence

API: (see also Integration of the motor control driver into other applications)

- *Set_speed()*
- *Set_ramp_down()*
- *Set_ramp_up()*
- *Get_req_speed()*
- *Get_speed()*
- *Get_status()*

The application source file **Demonstration.c** contains the following software routine:

- *Demonstration()* - Only for demonstration of the demo with a button. A sequence of speed changes and ramp changes for demonstration.

The application source file **peripheral.c** contains the following software routines:

- *PIT_init()* - Periodic interrupt timer init.
- *INT_init()* - Interrupt initialization.
- *FTM0_init()* - Flex Timer 0 initialization for 3-phase PWM signal generation.
- *FTM1_init()* - Flex Timer 1 initialization for speed measure.
- *SPI_init()* - SPI module for communication with a driver.
- *GPIO_init()* - GPIO initialization.
- *MC33927Config()* - Configuration of driver.
- *ADCinit()* - Analogue converters initialization.
- *PDBinit()* - Programmable delay block initialization.

The application source file **MC33927.c** contains the configuration of the 3-phase MOSFET driver. The application header file **main.h** includes the following application constants:

Table 8-1. main.h application constants

FMSTR_UART_PORT	*UART3_BASE_PTR*	FreeMASTER port set-up
FMSTR_UART_VECTOR	*67*	FreeMASTER interrupt vector set-up
FMSTR_UART_BAUD	*19200*	FreeMASTER speed set-up
CORE_CLK_KHZ	*48000*	Core clock set-up

The application header file **BLDC.h** includes the following application constants (See BLDC motor parameters (**BLDC.h**)):

- *FULL_DUTY*
- *HALF_DUTY*
- *MIN/MAX_CW/CCW_SPEED FRAC16*
- *MIN/MAX_MES_CW/CCW_SPEED FRAC16*
- *MIN_CW/CCW_SPEED_32*
- *RAMP_SCALE_CONST*
- *SPEED_TO_RPM_SCALE*
- *SCALE_CONST*
- *PHASE_A/B/C_FALLING/RISING*
- *SKIP_PWM_CYCLE*
- *STAND_THRESHOLD*
- *ENABLE/DISABLE_PWM_OUTPUT_PADS*
- *ALIGNMENT_CYCLE*
- *MIN_DC_BUS*
- *MAX_DC_BUS*
- *OVER_CURRENT_TIME_THRESHOLD*

Application Status:

- *IDLE = 0*
- *STOP = 1*
- *RUN = 2*
- *ALIGNMENT = 3*
- *EMERGENCY_STOP = 6*
- *UNDER_VOLTAGE_FAULT = 7*
- *OVER_VOLTAGE_FAULT = 8*
- *OVER_CURRENT_FAULT = 9*

The application header file **BLDC_config.h** includes the following application constants (See BLDC motor parameters (**BLDC_config.h**)):

- *PP*

- *TPM_C*
- *MAX_SCALED_SPEED*
- *PWM_FREQ*
- *BEMF_THRESHOLD*
- *START_DUTY_CYCLE*
- *SPEED_RAMP_UP*
- *SPEED_RAMP_DOWN*
- *PI_PROP_GAIN*
- *PI_INTEG_GAIN*
- *PI_INTEG_GAIN_SHIFT*
- *PI_INTEG_PART*

The application header file **freemaster_cfg.h** includes FreeMASTER settings.

The application header file **MC33927.h** includes MOSFET driver settings.

8.2 Microcontroller memory usage

Table 8-2 shows how much memory is needed to run the 3-phase BLDC sensorless drive (without FreeMaster and MQX) using the PK60N512VMD100 microcontroller. A significant part of the microcontroller memory is still available for other tasks.

Table 8-2. Memory usage

Memory	Available K60N512	Used
FLASH	512 KB	8.5 KB
RAM	128 KB	0.4 KB

8.3 Peripherals usage

For the proper functioning of this application, the following peripherals must be used. It is not allowed to use these peripherals for any other purpose.

Table 8-3. Peripherals usage

Module	Purpose
Timer PIT 0	• Periodic call of the speed control loop, application state machine, speed ramp, etc. • Period of an interrupt is 1.04 ms at 48 MHz core clock
Flex Timer FTM 0	• Generate a PWM signal • Run in combine mode • Switching frequency is 16 kHz on a 48 MHz core clock • Dead time 800 ms

Table continues on the next page...

Table 8-3. Peripherals usage (continued)

Module	Purpose
Flex Timer FTM 1	• Speed measurement • Prescaler is set to 128 • Modulo is 0xFFFF • Period of overflow is 175 ms
ADC 0	• Channel 19 - Used for the BEMF phase C sensing
ADC 1	• Channel 19 - Used for the BEMF phase A sensing • Channel 10 - Used for the DCB voltage sensing • Channel 0 - Used for the BEMF phase B sensing • Channel 12 - Used for the DCB current sensing
SPI 2	• Communication with the MC33937 MOSFET predriver
Ethernet	• Web control
SCI 0	• FreeMASTER interface
PDB 0	• ADC0 and the ADC1 to PWM synchronization
PTA 10	• Used to indicate the first level of overcurrent on the MC33937 MOSFET predriver
PTA 11	• Used to indicate FAULT mode
PTA 19	• Used for the emergency stop button
PTA 27	• Used to read the overcurrent pin on the MC33937
PTA 28	• Used to indicate the demonstration of the application
PTA 29	• Used to indicate RUN mode
PTE 26	• Used for the demonstration button

Chapter 9 References

9.1 References

1. *3-Phase BLDC/PMSM Low-Voltage Motor Control Drive User's Manual*, LVMCDBLDCPMSMUG, Initial version, by Freescale Semiconductor, Inc., 2009.
2. *K60 Sub-Family Reference Manual*. K60P144M100SF2RM, by Freescale Semiconductor, Inc., 2011.
3. *Set of General Math and Motor Control Functions for Cortex M4 Core*, MCLIBCORETXM4UG, by Freescale Semiconductor, Inc., 2011.
4. *3-Phase BLDC Motor Sensorless Control using MC9S08AW60*, DRM086, by Freescale Semiconductor, Inc., 2005.
5. *3-Phase BLDC Motor Sensorless Control using MC9S08MP16*, DRM117, by Freescale Semiconductor, Inc., 2009.
6. *3-Phase BLDC Sensorless Control using MCF51AG128*, DRM123, by Freescale Semiconductor, Inc., 2011.
7. *BLDC Motor Control with Hall Effect Sensors Using MQX on Kinetis*, AN4376, by Freescale Semiconductor, Inc., 2011.

For the above list of documentation, see http://www.freescale.com

9.2 Acronyms and abbreviations

Table 9-1. Acronyms

Term	Meaning
AC	Alternating current
ADC	Analog-to-digital converter
API	Application interface
ASM	Application state machine
BDM	Background debug mode
BEMF	Back-electromotive force

Table continues on the next page...

Table 9-1. Acronyms (continued)

Term	Meaning
BLDC	Brushless DC motor
CCW	Counter-clockwise direction
CW	Clockwise direction
DAC	Digital-to-analog converter
DC	Direct current
DMA	Direct memory access module
DRM	Design reference manual
DT	Dead time: a short time that must be inserted between the turning off of one transistor in the inverter half bridge and turning on of the complementary transistor because of the limited switching speed of the transistors
FTM	FlexTimer module
GPIO	General-purpose input/output
I/O	Input/output interfaces between a computer system and the external world — a CPU reads an input to sense the level of an external signal and writes to an output to change the level of an external signal
ISR	Interrupt Service Routine
PK60N512	Freescale 32-bit ARM based microcontroller
MCU	Microcontroller
PDB	Programmable delay block module
POSIX	Portable Operating System Interface, produced by IEEE and standardized by ANSI and ISO. MQX conforms to POSIX.4 (real-time extensions), and POSIX.4a (threads extensions)
PWM	Pulse-width modulation
RPM	Revolutions per minute
RTOS	Real Time Operating System
RTCS	Embedded Internet stack provides IP networking for the MQX platform. RTCS is provided with a rich assortment of TCP/IP networking application protocols and uses the MQX RTOS drivers for Ethernet or serial connectivity
SCI	Serial communication interface module: a module that supports asynchronous communication
SPI	Serial peripheral interface module

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: DRM135
Rev. 0, 06/2012